МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ

імені ІГОРЯ СІКОРСЬКОГО»

ПРОГРАМУВАННЯ МОВОЮ С: ІНСТРУКЦІЇ ДО ВИКОНАННЯ ЛАБОРАТОРНИХ РОБІТ З ДИСЦИПЛІНИ «ПРОГРАМУВАННЯ-2. ПРОГРАМУВАННЯ МОВОЮ С»

Рекомендовано Методичною радою КПІ ім. Ігоря Сікорського

як навчальний посібник для студентів,

які навчаються за спеціальністю 123 Комп'ютерна інженерія

Київ

КПІ ім. Ігоря Сікорського

2021

Рецензенти: *Онай М.В., к.т.н., доцент.*
Мальчиков В.В., ст.викладач.

Відповідальний редактор *Тарасенко В.П., д.т.н.,професор.*

Гриф надано Методичною радою КПІ ім. Ігоря Сікорського
(протокол № 6 від 25.02.2021 р.)
за поданням Вченої ради факультету прикладної математики
(протокол №8 від 27.01.2021 р.)
Електронне мережне навчальне видання
Романкевич Віталій Олексійович, д.т.н., професор
Тарасенко-Клятченко Оксана Володимирівна, к.т.н., доцент
Клятченко Ярослав Михайлович, к.т.н., доцент

ПРОГРАМУВАННЯ МОВОЮ С: ІНСТРУКЦІЇ ДО ВИКОНАННЯ ЛАБОРАТОРНИХ РОБІТ З ДИСЦИПЛІНИ «ПРОГРАМУВАННЯ-2. ПРОГРАМУВАННЯ МОВОЮ С»

Програмування мовою С: інструкції до виконання лабораторних робіт з дисципліни «Програмування-2. Програмування мовою С» [Електронний ресурс] : навч. посіб. для студ. спеціальності 123 Комп'ютерна інженерія / КПІ ім. Ігоря Сікорського; В.О. Романкевич, О.В. Тарасенко-Клятченко, Я.М. Клятченко, – Електронні текстові дані (1 файл: 2,8 Мбайт). – Київ: КПІ ім. Ігоря Сікорського, 2021. – 142 с.

Навчальний посібник надає теоретичні відомості, які необхідні для виконання лабораторних робіт, докладні приклади на кожну тему, варіанти завдань на лабораторні роботи, контрольні запитання, відповідаючи на які студент може перевірити свої знання теоретичного та практичного матеріалу, рекомендовану літературу, яка дозволяє ширше вивчити теоретичний матеріал з тем курсу. Навчальне видання призначене для студентів, які навчаються за спеціальністю 123 «Комп'ютерна інженерія» факультету прикладної математики НТУУ «КПІ ім. Ігоря Сікорського».

ЗМІСТ

1. ОГЛЯД МОВИ С. ЕЛЕМЕНТИ МОВИ

Історія мови С тісно пов'язана з появою операційної системи Unix, яку було розроблено 1969 р. у фірмі Bell Labs для міні-ЕОМ. На початку 60-х рр. створено мову програмування Алгол 60, потім у Кембриджі 1963 і 1967 рр.- мови CPL (Combined Programming Language) та BCPL (Basic CPL). Компілятор для мови програмування В розробив Кен Томпсон 1970р. у фірмі Bell Labs, мову програмування С - Денніс Рітчі 1972 р. також у фірмі Bell Labs. Далі розвиток відбувався у напрямку створення компіляторів для удосконалених мов програмування С++ та С#.

Перетворення вихідної програми, написаної мовою С, у код виконання (файл *.exe) відбувається у три етапи: препроцесування, компіляція і компоновка. Препроцесор підключає зовнішні файли і розширює макроси. Компілятор транслює результати роботи препроцесора в об'єктний код (файл *.obj), якщо в тексті програми немає синтаксичних і семантичних помилок. Компоновщик зв'язує об'єктний код, створений компілятором, із програмами з бібліотек або іншими файлами. Результатом компоновки є exe-файл.

Програма мовою С — це рядок символів, що складається з лексичних елементів п'яти типів: ключові (зарезервовані) слова, константи, операції, розділювачі та ідентифікатори. Розділювачі відокремлюють суміжні елементи і являють собою пробіли, символи табуляції та переведення рядка/повернення каретки. Коментар - це текст, що знаходиться між символами /* і */ для багаторядкового коментаря і після символів // для однорядкового коментаря. Такий текст компілятором ігнорується. Коментарі служать для документування і полегшення налагодження програмних продуктів. Коментар може містити будь-яку кількість будь-яких символів та рядків і сприймається компілятором, як розділювач.

Застереження: помилкою буде вважати, що за кодом програми, який не містить коментарів, можна легко зрозуміти зміст програми (навіть її розробнику).

Ідентифікатори служать для найменування типів даних, змінних, констант і функцій. Ідентифікатор включає літери, цифри і знак підкреслювання, не може починатися з цифри і збігатися із зарезервованим словом. В ідентифікаторах розрізняються велика й малі літери, тому ідентифікатори count і COUNT є різними.

Використання раціональних ідентифікаторів змінних та імен функцій полегшує коментування і читання програмного коду (наприклад, доцільно вживати смислові назви: int iCounter, VectorSum()).

Зарезервованими словами (такими як break, char, const, do, float, int. return, sizeof, struct, while та ін.) не можна користуватися як ідентифікаторами. Список зарезервованих слыв наведено в додатку 3.

Константи бувають чотирьох типів: цілі, дійсні, символьні і рядкові. Константи цілого типу — це числа у вісімковій, десятковій або шістнадцятковій системах числення. Десяткові константи складаються з цифр, причому першою не має бути нуль, вісімкові починаються з нуля і містять цифри від 0 до 7, шістянадцяткові починаються з нуля, потім слідує x або X, а далі наводяться шістнадцяткові символи (0..9, A..F).

Вважається, що значення цілої константи завжди додатнє. Знак «-» перед константою компілятор сприймає, як операцію зміни знака, а не як частину константи (тобто: від'ємне_число = (додатня_константа)*(-1)).

Константи з плаваючою точкою - де цифри, десяткова точка і знаки десяткового порядку e або E, наприклад: «1.», «1.72», «2e1», «0.35e-б».

Вважається, що аналогічно цілим константам значення дійсних

констант завжди додатнє, тобто «-» — це символ унарної операції зміни знака.

Символьні константи — це символи, обмежені одинарними лапками. Такі константи мають тип int і збігаються з кодами символів ASCII. Рекомендується користуватися символьними константами, а не їх цифровими кодами (наприклад, ch='a' краще, ніж ch=97). Керуючі коди (ескейп-послідовності) також беруть у лапки, наприклад: '\n' - символ переведення рядка, '\t' -горизонтальна табуляція. '\ddd' — вісімковий код символу ASCII. Рядкові константи беруть у подвійні лапки:

str="Приклад тексту \n що складається з двох рядків \n\n";

Типом даних визначаються набір дозволених значень і набір операцій, які можна виконувати зі змінними цього типу. Кожен тип даних (у кожній реалізації мови С) займає певну ємність пам'яті (char -1 байт, int - 2 або 4 байт і т. д.). Ємність пам'яті для елементів можна обчислити за допомогою операції sizeof.

Перед використанням змінних цілого типу їх треба описати специфікатором відповідного типу, наприклад: int age; float pi=3.14; char c='z';. Якщо застосовано специфікатор unsigned, long або short, то специфікатор int можна не наводити.

short int	int	long int	unsigned short int	unsigned long int
2 байти	4 байти	4 байти	2 байти	4 байти
–32768 ..32767	–2147483648 .. 2147483647	–2147483648 .. 2147483647	0 ..65536	0 .. 4 294 967 295

char	signed char	unsigned char
1 байт	1 байт	1 байт
-128..127	0..255	-128..127
float	double	long double
4 байта	8 байтів	10
3.4e-38..3.4e+38	1.7e-308..1.7e+308	3.4e-4932..1.1e+4932

До основних типів відноситься також тип void. Множина значень цього типа - пусто. Не можна описати величину типа void. Використовується при роботі з функціями, вказівниками.

Операції над даними типу int:

- арифметичні: +.-,*, /, %, ++ (инкремент), - - (декремент);
- логічні: && (AND), || (OR),! (NOT), = = (порівняння);
- бітові: & (AND). I (OR), ^ (XOR), - (NOT), » (правий зсув), «(лівий зсув).

Змінні дійсного типу мають таку форму запису: float sum=17.85. Пам'ять для дійсних змінних у 2 (або 4) рази більша, ніж для даних типу int. Тому відповідні арифметичні операції виконуються повільніше. Набір операцій той самий, що й для даних типу int (за винятком бітових операцій).

Символьні змінні - це змінні типу char, які займають 1 байт. За замовчанням символьні змінні мають знак. Беззнакові символьні змінні записують таким чином: unsigned char, і вони обмежені діапазоном 0-255 Оскільки тип даних char - це цілий тип, то ним можна користуватися також разом з типом даних int, наприклад: int a=3+'z';

Пріоритет і порядок виконання операцій. Порядок виконання операцій визначається порядком круглих дужок. Якщо запис не містить дужок, то операції виконуються з урахуванням їхніх пріоритетів. Пріоритет операцій і їх асоціативність наведено в додатку 4.

Перераховуваний тип даних. Дає можливість програмісту створити свій тип даних.

Приклад:

enum(red, amber, green} traffic_light;

Кожній константі перераховуваного зліченного типу ставлять у відповідність деяке ціле число, наприклад: red=0, amber=l, green=2. Числа можуть бути довільними: enum traffic_light{red=7, amber=5, green=3};.

Звертатися до змінних можна наступним чином:
if(tfaffic_light = = 5) або if(traffic_light = = amber).

Перетворення типів має такий синтаксис: (ім'я_типу) вираз.
Наприклад: float r=3.5; int i; i=(int)r;

В арифметичних операціях перетворення типів виконується автоматично. Форма запису операції присвоювання передбачає, що ліва і права частини запису є однотипними. Якщо така умова не виконується, то компілятор намагається звести тип правої частини до типу лівої. Наприклад: i=(float) 10. При цьому ціле значення 10 зводиться до типу float 10.0.

Автоматичне перетворення типів здійснюється в унарних операціях !,-,-,* (логічне NOT, унарний мінус (від'ємне число), бітове заперечення, звернення за адресою) і в операціях зсуву » та «.

У разі виконання двомісних операцій обом операндам присвоюють простий загальний тип і одержують результат такого самого типу:

```
int    i=5;
double      r=7.3;
if(i<r)
…
```

У цьому випадку змінній і, яка дорівнює 5, перед порівнянням зі змінною r присвоюється форма 5.0 (тип double).

2. ОРГАНІЗАЦІЯ ВВЕДЕННЯ-ВИВЕДЕННЯ

Перш за все слід розглянути виведення даних, оскільки програма, в якій відсутнє виведення інформації, навряд чи зможе бути корисною. Як правило, виведення полягає у записі інформації (слів та зображень) на екран, у накопичувач (гнучкий або жорсткий диск) або порт вводу/виводу (послідовний порт, порт принтера тощо).

Функція printf(). Функція printf(), що найчастіше використовується для виведення інформації, призначена для запису інформації на екран і має наступний:

printf (<рядок-формату>,<елемент>,<елемент>,…);

Рядок формату – це рядок, що починається та закінчується подвійними лапками. Функція printf() виводить рядок у стандартний вивід (stdout). При цьому printf() підставляє у рядок перелічені <елементи> відповідно до команд форматування, наведених у рядку. Наприклад, в попередній програмі мав місце наступний оператор printf():

printf(“Сума дорівнює %d \n”, sum);

%d – одна з команд форматування, що називається специфікацією формату. Всі специфікації формату починаються з символу процента %, після якого вказується буква, яка задає тип даних та тип форматування виразу, що підставляється.

Кожній специфікації формату відповідає рівно один елемент зі списку. Якщо тип даних елемента не відповідає специфікації формату, можна отримати непередбачуваний результат. Власне елементами можуть бути змінні, константи, вирази, виклики функцій, тобто будь-яке значення, що допускається відповідною специфікацією формату.

Специфікація формату %d означає, що виводитиметься ціле значення.

Далі наведені інші часто використовувані специфікації форматів.

%u	беззнакове ціле
%ld	довге ціле
%p	значення вказівника
%f	плаваюча крапка
%e	плаваюча крапка з експонентою
%c	символ
%s	рядок
%x або %X	ціле у шістнадцятковому форматі

Ширину поля виводу, у якому видаються дані, можна встановити за допомогою числа, що вказується між % та буквою специфікації формату; наприклад, для виводу цілого значення в полі з шириною у 4 символи необхідно ввести %4d.

Якщо необхідно вивести знак проценту, слід вказати %%.

Символи \n не є специфікацією формату. За історичними причинами ці спеціальні символи називають escape-послідовністю. У даному випадку \n означає, що вставлено символ нового рядка, який після виводу рядка переміщує курсор на початок нового рядка.

Найбільш часто вживані наступні:

\f	переведення сторінки
\t	табуляція
\b	повернення назад на одну позицію
\x <hhh>	вставка символу, заданого ASCII-кодом <hhh>, де <hhh> – це 1-3 шістнадцятирічні цифри

Для виводу зворотного слешу (похилої риски) слід вставити комбінацію \\.

Функції виведення puts() та putchar(). Функція puts() виводить на екран рядок, додаючи до нього символ нового рядка.

Наприклад, просту програму можна переписати у наступному вигляді:

```
#include <stdio.h>
main()        {
        puts("Привіт, світе!");
}
```

Варто відзначити, що у кінці рядка відсутня комбінація \n, оскільки функція puts автоматично додає її до рядка.

Функція putchar виводить на екран один символ без додання \n. Оператор putchar(ch) еквівалентний printf("%c",ch).

Одна з причин використання функцій puts() та/або putchar() замість функції printf() полягає в тому, що функція, яка звертається до printf(), займає багато пам'яті, тому printf() рекомендується використовувати лише для форматування чисел або спеціального форматування, а в інших випадках звертатися до puts() та putchar(). Тоді не лише зменшиться розмір програми, а й прискориться її виконання. Наприклад, файл .exe, створений при компіляції версії програми, що використовує puts(), значно менший за версію з printf().

Функція scanf(). Для інтерактивного вводу даних в основному використовується scanf(), що має багато спільного з printf(). Функція scanf() має наступний формат:

```
scanf(<рядок-формату>,<адреса>,<адреса>,…);
```

У scanf() використовуються такі ж специфікатори формату,як і в printf(): для цілих – %d, для значень з плаваючою крапкою – %f, для рядків – %s тощо.

Проте існує одна суттєва відмінність між scanf() та printf(): елементи, вказані після рядка формату, мають бути адресами, а не значеннями. Виклик вигляду

scanf(“%d %d”, &a, &b);

означає, що очікується введення з клавіатури двох десяткових (цілих) значень, розділених проміжком; перше значення буде збережено за адресою, пов‘язаною з a, друге – за адресою, пов‘язаною з b. Адреси a та b в scanf() тут передаються за допомогою оператора прямої адресації (&).

Фактично проміжком між двома специфікаторами формату %d може бути не лише один пробіл, а й довільне число пробільних символів (тобто будь-яка комбінація пробілів, символів табуляції та символів нового рядка).

Однак якщо замість пробілу розділити числа комою, то оператор вигляду

scanf(“%d,%d”, &a, &b);

дозолить вводити значення, розділені комою.

Передача адреси у функцію scanf(). Введення рядків можна розглянути на прикладі наступної програми:

```
#include <stdio.h>
main() {
char name[150];
printf(“Ваше ім‘я:”);
scanf(“%s”, name);
printf(“Привіт, %s\n”, name);
}
```

Оскільки змінна name є масивом символів, її значенням є адреса масиву. Тому перед name не потрібно вказувати оператор &; достатньо вказати scanf(“%s”, name).

Слід відзначити, що використовувався масив (char name[150];), а не вказівник (char *name), оскільки при описі масиву виділяється пам‘ять для рядка, а при описі вказівника цього не відбувається. При використанні char *name необхідно явно виділити пам‘ять для name.

Використання функцій gets() та getch() для введення. При використанні scanf() для введення рядків виникає інша проблема, яку можна продемонструвати, якщо знову виконати програму, але ввести повне ім‘я. Видно, що програма видає лише прізвище, оскільки введений після нього пробіл для scanf() означає кінець введеного рядка.

Існують два можливі вирішення цієї проблеми. Одне з них показано у наступній програмі:

```
#include <stdio.h>
main(){
char first[30], middle[30], last[30];
printf(“Ваше ім‘я:”);
scanf(“%s %s %s”, first, middle, last);
printf(“Привіт, любий %s, чи слід казати %s?\n”, last, first);
}
```

У цьому прикладі передбачається введення по-батькові; функція scanf() очікує введення трьох рядків. Проте це не вирішує проблеми вводу в одному рядку повного імені з пробілами.

Нижче наведено друге рішення:

```
#include <stdio.h>
main(){
char name[150];
printf(“Ваше ім‘я:”);
gets(“%s”, name);
printf(“Привіт, %s\n”, name);
}
```

Функція gets() зчитує введенні дані до натиснення Enter. При цьому відповідно Enter не запам‘ятовується у рядку, але в кінець рядка додається нульовий символ (\0). І, нарешті, існує функція getch() для зчитування з клавіатури одного символа, причому символ, що вводиться, не відображається на екрані (на відміну від scanf() та gets()). Слід вказати, що getch не приймає параметр-символ, але є функцією, що повертає значення з типом char.

Потоки та введення/виведення потоком. Потоки є найбільш мобільним засобом С для читання та запису даних, який забезпечує гнучке та ефективне введення/виведення і не залежить від файлів та апаратних засобів.

Потік – це файл або фізичний пристрій (наприклад, принтер або монітор). Для роботи з потоком використовується вказівник на об‘єкт FILE (визначений у stdio.h). Об‘єкт FILE містить інформацію про потік, включаючи поточне положення у потоці, вказівники на відповідні буфери, а також індикатори помилки та кінця файлу.

Власне програма не повинна створювати та копіювати об‘єкти FILE. Замість цього програма використовує вказівники, що повертаються функціями типу

fopen(). Не слід змішувати вказівники FILE з обробниками файлів у DOS (використовуються в DOS при операціях введення/виведення) низького рівня, а також операціях введення/виведення, сумісних з UNIX).

Перед виконанням введення/виведення спочатку необхідно відкрити потік, що встановлює з'єднання потоку з файлом або пристроєм, які мають ім'я DOS. Для відкриття потоку використовуються функції fopen() та freopen(). При відкритті потоку необхідно задати режим роботи: читання, запис або читання та запис. Також слід вказати спосіб обробки даних потоку: текст або двійкові дані. Останнє є суттєвим через деякі несумісності між введенням/виведенням потоку С та текстовими файлами DOS.

Текстові та двійкові потоки. Текстові потоки використовуються для роботи зі звичайними текстовими файлами DOS. При введенні/виведення потоком в С передбачається, що текстові файли розділені на рядки за допомогою одного символу нового рядка (відповідного коду ASCII для переведення рядка). Проте у текстових файлах DOS, що зберігаються на диску, сусідні рядки розділені двома символами: символом повернення каретки та символом переведення рядка. В текстовому режимі Turbo C під час введення перетворює пари символів повернення каретки/переведення рядка (CR/LF) в один символ переведення рядка; при виведенні символ переведення рядка замінюється на пару CR/LF.

Двійкові потоки набагато простіші, ніж текстові потоки, оскільки в них відсутні вказані перетворення. Всі символи читаються та записуються без змін.

До файлу можна звертатися як у текстовому, так і в двійковому режимі. Розуміння перетворень, що виконуються у текстових потоках, дозволяє уникнути можливих проблем. Сі не «запам'ятовує» дату та час створення файлу й останнього звертання.

Якщо при відкритті потоку не задано режим перетворення, потік відкривається у режимі перетворення за умовчанням, який визначається значенням глобальної змінної _fmode. За умовчанням для _fmode встановлено текстовий режим.

Буферизація потоків. Асоційовані з файлами потоки є буферизованими, що дозволяє з високою швидкістю виконувати введення/виведення на рівні окремих символів функціями типу getc() та putc(). Програміст за допомогою викликів setvbuf() та setbuf() може задавати буфер, змінювати розмір буфера, що використовується, а також скасовувати використання буфера потоком.

Заповнення буфера, закриття потоку та нормальне завершення програми викликають автоматичне очищення буфера. Так само для очищення буферів можна використовувати функції fflush() і flushall().

Потоки використовуються для послідовного читання та запису даних. При цьому введення/виведення здійснюється для поточного положення у файлі. Під час читання та запису даних програма встановлює вказівник положення у файлі безпосередньо після даних, до яких відбувалося останнє звертання. Але потоки, з'єднані з файлом на диску, підтримують прямий доступ: за допомогою команди fseek встановлюється положення у файлі, а потім для звернення до даних, розташованих після поточного положення у файлі, виконується кілька операцій читання або запису.

Якщо потік відкритий для читання чи запису даних, існують обмеження на використання операцій читання й запису. Між читанням і записом буфер потоку слід звільняти викликом fflush(), flushall() або fseek(). Для максимальної мобільності програми буфер необхідно звільняти навіть за його відсутності, оскільки в інших системах навіть у небуферизованних потоках можуть існувати додаткові обмеження на змішування операцій введення та виведення.

Попередньо визначені (стандартні) потоки. На додаток до потоків, які створюються викликом fopen(), існує п'ять попередньо визначених потоків, доступних з моменту початку виконання програми.

Ім'я	Введення/ виведення	Режим	Потік
stdin	Введення	Текстовий	Стандартне введення
stdout	Виведення	Текстовий	Стандартне виведення
stderr	Виведення	Текстовий	Стандартна помилка
stdaux	Введення та виведення	Двійковий	Додаткове введення/виведення
stdprn	Виведення	Двійковий	Виведення на принтер

Потоки stdaux і stdprn є специфічними для DOS, і тому не переносяться в інші системи.

Потоки stdin і stdout можна переключати в DOS, в той час як інші потоки з'єднані (зв’язані?) з певними пристроями: stderr — з консоллю (CON:), stdprn — із принтером (PRN:), а stdaux — з додатковим портом.

Додатковий порт визначається конкретною конфігурацією комп'ютера; зазвичай таким портом є COM1. Відомості про перемикання введення/виведення за допомогою командного рядка DOS див. у посібнику з DOS. Якщо перемикання введення/виводу відсутнє, потоки stdin і stdout з'єднуються з консоллю (пристроєм CON:). Крім того, за відсутності перемикання в stdin використовуються буферизовані рядки, а stdout — небуферизований потік. Решта попередньо визначених потоків є небуферизованими.

Для обробки попередньо визначеного потоку в режимі, відмінному від режиму за умовчанням, наприклад, для обробки stdprn у текстовому режимі,

використовується функція setmode. Імена попередньо визначених потоків є константами; присвоєння імен таким потокам не допускається. Для з'єднання визначеного потоку з іншим файлом або пристроєм використовується freopen.

3. КЕРУЮЧІ ОПЕРАТОРИ

Складений оператор (блок) - це послідовність операторів, обмежених фігурними дужками. Кожний оператор блоку закінчується крапкою з комою, після складеного оператора крапку з комою не ставлять.

Порожній оператор - це крапка з комою, перед якою немає виразу. Такий оператор використовують у циклі, всі дії якого вже описані в заголовку. Рекомендується порожній оператор розміщувати на окремому рядку для полегшення аналізу програмного коду.

Керуючі структури бувають двох видів: структури вибору і структури циклу. У структурах вибору круглі дужки обмежують усі логічні умови конструкції вибору.

Оператор if має синтаксис:

```
if(вираз)     оператор;
```

Оператор конструкції if може бути складений, простий або порожній Якщо вираз у заголовку умовного оператора істинний (не нульовий) то виконується оператор умовної конструкції, якщо помилковий (false, нуль) то керування передається оператору, який слідує після конструкції вибору, наприклад:

```
if(x+y-z)>=w
{
        x+=3; -=4; z*=6;
}
        w/=2;
```

Синтаксис конструкції вибору if...else:

```
if (вираз)    оператор_1
else     оператор_2
```

Якщо значення виразу (умови) не дорівнює нулю (true), то виконується оператор_1, в іншому випадку виконується оператор_2:

```
int i=4,j=6, k=8; if(i<k)
```

```
    if(j<i)
        printf("one\n");
else
        printf("two\n");
```

Оскільки i<k - true, а j<i - false, виникає думка, що виводу взагалі не буде. Але за правилами синтаксису else належить найближчому оператору if, тому буде виведено two.

Умовною тернарною операцією «?:» користуються замість конструкції if…else за умови, що оператори, які входять до цієї конструкції, є простими виразами.

Синтаксис умовної операції: результат=вираз_1 ? вираз_2 : вираз_3.

Наприклад:

```
int i=6;
    j=4;
int res=(i<j) ? i: j;
printf("%d\n",res);
```

Після виконання цієї послідовності операторів буде виведено 4 (тобто зміна res має значення j).

Оператором switch можна замінити складний оператор розгалуження if…else:

```
switch(вираз)
{
case константа_1:
        оператор(и)
case константа_2:
        оператор(и)
default
        оператор(и)
}
```

Послідовність дій під час виконання оператора switch:

а) Обчислюється вираз у заголовку;

б) Результат послідовно порівнюється з усіма константними виразами;

в) Якщо значення збігаються, то виконуються відповідні оператори і продовжується (тому кожна послідовність case має містити оператор break для виходу з конструкції switch);

г) якщо значення не збіглося, виконується гілка default (замовчання). Але цієї гілки в конструкції switch може і не бути, наприклад, код помилки містить деяка змінна int_err (можливі значення кодів помилок – від 1 до 3).

Тоді:

```
switch(err)
 {
 case1: printf("помилка_1\n"); break;
 case2: printf("помилка_2\n"); break;
 case3: printf("помилка_3\n"); break;

 default: print ("недійсний код помилки");
}
```

Оператор goto використовують для безумовної передачі керування всередині функції від одного оператор а на інший. Форма запису

goto label_A;

означає передачу керування на оператор, позначений у тій самій функції ідентифікатором label_A.

Оператором goto користуватися не рекомендується через можливі ускладнення супроводу програмного продукта і погіршення можливостей оптимізації програми компілятором.

Цикл while має таку форму запису:

```
while(умова)
        оператор(и)
```

Умова перевіряється до виконання оператора (ів) тіла циклу. Якщо умова - true, то виконується оператор (група операторів). Щоб уникнути зациклення, в тілі циклу потрібно змінювати змінні, що входять до умовного виразу. Типовою помилкою є такий запис умови, за якого виконання циклу ніколи не припиняється (виконується так званий нескінченний цикл), наприклад:

```
int i=1;
while (i>0)
i++;
```

Цикл while припиняє працювати в таких випадках;

1) Умова в заголовку набула нульового значення;
2) У циклі виконано оператор break;
3) У циклі виконано оператор return.

У першому і другому випадках керування передається оператору, що йде за циклом while, у третьому випадку відбувається вихід з функції.

Помилковою є така форма запису циклу while:

```
int i=1;
while(i=5)
i++;
```

Це нескінченний цикл, але компілятор видасть відповідне попередження. Тут помилково використано операцію присвоєння =, тоді як потрібно було записати while(i==5).

Також можлива помилка типу «зайвий крок»:

```
int data[100];
Size=0;
```

```
While(size<=100)
Data[size++]=size;
```

Цикл буде виконано 101 раз. Спроба звернення до елемента масиву з індексом 100 (тобто до 101-го елемента) може виявитися небезпечною.

Форма циклу do…while передбачає перевірку умови виконання оператора або тіла циклу:

```
do
        оператор
while (умова)
```

Оператор циклу може бути простим або складеним. Вихід із циклу do…while відбувається за умов, аналогічних умовам входу з циклу while.

Цикл for – це найуживаніший цикл в мові С:

for([вираз_1]; [вираз_2]; [вираз_3];) оператор(и);

Кожен із трьох виразів можна випускати. Зміст виразів може бути будь – який, але, як правило, перший вираз - це ініціалізація циклу, другий – перевірка умови закінчення циклу, третій – зміна керуючої змінної. Можна провести аналогію з циклом while:

```
вираз_1;
while(вираз_2);
{
        оператор(и);
        вираз_3;
}
```

Примітка. Форма запису циклу for(;;); є синтаксично коректною, але цикл виконуватиметься нескінченно.

Приклад організації друку всіх парних чисел у діапазоні 1000000…0;

```
long I ;
for(i=1000000; i>=0; printf(“%ld\n”,i)), i-=2)
```

Формальний алгоритм виконання циклу for:

1) якщо є вираз_1, то він обчислюється;
2) якщо є вираз_2, то він також обчислюється (якщо значення оператора "вираз_2"=false (нуль), то виконання циклу припиняється, якщо true (!=0), то роботу з циклом буде продовжено);
3) обчислюються оператори тіла циклу;
4) обчислюється вираз_3 (якщо він є);
5) перехід до п. 2.

Поява оператора continue в будь-якому місці тіла циклу зумовлює негайний перехід до п. 4 (тобто до обчислення оператора "вираз_3").

4. МАСИВИ, РЯДКИ ТА ВКАЗІВНИКИ

Масив – це сукупність однотипних даних.

Масив в програмі описується наступним чином:

(тип елементів) (ім’я масива) [(кількість елементів масива)]

При цьому виділяється пам'ять об’ємом (sizeof(тип) * розмір_масиву) байтів. Масиви індексують, починаючи з нуля, наприклад, масив float data[128] містить 128 елементів типу float: data[0], data[1], ... data[127]. Ім'я масиву - це константа-вказівник на початкову адресу даних, тобто на нульовий елемент масиву.

Опис char*str - це не тільки вказівник на символьну змінну, це ще й опис масиву-рядка з початковою адресою str і невизначеною кількістю елементів. Неініціалізований вказівник (адресу) можна ініціалізувати, описавши йото статично (тобто, як static) або отримавши адресу пам'яті за допомогою функції malloc(). Арифметичні операції для вказівників - це всі операції цілочисельної арифметики. Наприклад, для масиву int data[100] адресу 100-го елемента можна отримати, як &data[99]. Але зробити те саме можна і так: data+99.

Адреси сусідніх елементів відрізняються на кількість байтів, необхідних для зберігання даних відповідного типу (наприклад, на 2, якщо sizeof(int)=2.

Серед помилок при роботі з масивами найбільш поширена - це вихід за межі масиву. Наслідком є неправильний результат і можливий збій системи. Відповідальність за перевірку виходу за межі масивів у мові С цілком покладено на програміста.

Ініціалізувати масив можна при його описанні:

(тип) імя_масиву []={ знач_1, знач_2,…, знач_n};

Розмір масиву в квадратних дужках можна при цьому не визначати:

```
int data[5]={ 10,20,30,40,50};
char str[]={'Р','я','д','о','к','\n'};
```

Якщо масив описано статично (тобто поза функцією), то він ініціалізується нульовим значенням (як і інші змінні з ознакою static):

```
Static int data[5];     /* усі 5 елементів - нулі*/
```

Масиви великої розмірності з ненульовими початковими значеннями ініціалізують у циклі:

```
for(i=0; i<size;i++) data[i]=value;
```

Символьні масиви (рядки) - це одновимірні масиви символів, які закінчуються символом '\0'. Рядок обмежують подвійними лапками.

Наприклад:

```
char mess1[20]="Just a string.";
char mess2[]="Just a string.";
char *mess3="Just a string.";
char mess4="Just a string.";
```

Під час копіювання рядків присвоювати адресу одного рядка адресі іншого забороняється. Але можна організувати такий цикл:

```
int i=0;
while(strl[i]=str2[i])
 i++;
```

Тоді копіювання закінчиться, коли str2 [i] буде дорівнювати '\0'. Для копіювання рядків можна обирати також стандартну функцію strcpy (strl,str2) з файлу string.h.

Типовою є помилка, пов'язана з відсутністю символа '\0' у кінці рядка. Якщо рядок задається літералом, то компілятор додасть символ ‘\0’ в кінці автоматично. Якщо рядок формується символ за символом, то за правильне завершення рядка відповідає програміст.

Передача масивів і вказівників як параметрів функцій. У функцію передається тільки початкова адреса масиву. Зміна елементів масиву всередині функції впливає на стан початкового масиву, але зазвичай параметри в функцію передаються за значенням, при цьому всередині функції створюється копія параметра, наприклад:

```
void print_table(char *name_tab, double *xtab, double *ytab, int numlines)
    {
    Printf(“\n\n Значення функції % s \n\n”, name_tab);
    While(numlines--)
    Printf(“%f, %14f\n”, *xtab++, *ytab++)
    }
```

Перший параметр функції print_table це рядок name з ім'ям таблиці. Другий і третій параметри - xtab і ytab - це початкові адреси двох масивів типу double. Четвертий параметр - кількість рядків таблиці numlines - передається за значенням, тобто всередині функції print_table буде створено копію змінної numlines і використано її, як лічильник циклу while.

Довільний масив може мати n вимірів.

Наприклад, описання тривимірного масива може бути таким:

```
int array [2][3][4];
```

У багатовимірних масивах найшвидше змінюється останній індекс.

Масив int ar[2][3] розміщується в пам’яті так:

Ar[0][0], Ar[0][1], Ar[0][2], Ar[1][0], Ar[1][1], Ar[1][2]

Багатовимірні масиви часто потребують виділення великих об’ємів пам’яті, наприклад, double data [50] [50][50] потребуватиме 1250000*sizeof(double) байт пам'яті. Тому при передачі масива в функцію доцільніше передавати не весь масив, а вказівник на нього.

Параметрами командного рядка користуються для передні інформації в програму (наприклад, main(int argc, char* argv[]) і відокремлюють ці параметри пробілами. Параметр argc завжди більший або дорівнює одиниці і містить кількість параметрів командного рядка. Параметр argv - це масив рядків довільної довжини (так званий вільний масив). Елемент argv[0] містить повне ім'я файлу програми, а елемент argv[1] зберігає значення argc (якщо воно існує).

Приклад: prog.exe file_name (file_name - ім'я файлу, який використовується програмою prog.exe; argc=2, argv[0] містить ім'я файлу (prog.exe), argv[1] містить рядок file_name).

Вільні масиви - це двовимірні масиви (матриці) з різною довжиною рядків:

char* names[]={ "Паниковский", "Балаганов", "Козлевич", "Бендер"};

Компілятор виділяє для вільного масиву потрібну кількість байтів, яка дорівнює сумі довжин рядків, плюс байти для символів ‘\0’

На елементи масиву рядків можна посилатися таким чином

Names[3] = «Бендер»;

Вказівники і масиви. Звернення до масиву компілятор інтерпретує, як операцію з вказівниками, наприклад ar[6] відповідає *(ar+6), ar[2][3] відповідає *(*(ar+2)+3).

Вказівник - це адреса пам'яті. Опис int*x компілятор сприймає, як "x є вказівником на ціле". Вказівник на тип даних void сумісний з будь-яким вказівником, наприклад, void *x;, int *y;. При цьому дозволено присвоювати

y=x. Операція * є операцією звернення за адресою (іноді ЇЇ називають "операцією розіменування"). Форма запису

*ptr_var=value;

означає процедуру знаходження значення value, при цьому саме значення зберігається за адресою ptr_var.

Операція визначення адреси & - це процедура одержання адреси операнда. Форма запису операції: (адреса)=&(змінна). Усім вказівникам мови С можна присвоювати безпечну адресу - нуль цілого типу. Гарантовано, що ця адреса не буде збігатися з жодною з адрес програми. Для розподілу адрес застосовують функцію виділення пам’яті malloc.

x=(int*)malloc(sizeof(int));

Завдяки застосуванню функції malloc() поліпшуються можливості перенесення (мобільності) програмних продуктів. прототип функції malloc знаходиться у файлі alloc.h. Функція malloc() вертає дані типу *void, тому слід виконати перетворення типу на *int, якщо змінну x описано десь вище, як вказівник на int. Якщо пам'яті не вистачає, то функція malloc вертає нуль, тому рекомендується робити перевірку:

```
if(x=int*)malloc(sizeof(int)))!=0)
{
  послідовність операторів
}
```

Можливе помилкове застосування вказівників. Наприклад, якщо змінну-вказівник описано всередині функції, то ця змінна не ініціалізується і пам'ять для її значення не виділяється. Можна описати вказівник поза функцією або надати їй властивість static (видимість у межах файлу). Тоді вказівник ініціалізується нулем і для нього виділяється потрібна пам'ять. Ще одна проблема - це незвільнення пам'яті, виділеної функцією malloc,

тоді, коли вказівник уже не потрібний. Вирішити її можна, використавши функцію free() з аргументом-вказівником на пам'ять, що звільняється.

Помилкою вважається присвоювання вказівникові конкретного значення адреси, наприклад int*x=123456. Компілятор може допустити таку ситуацію, але мобільність програми при цьому зменшиться. Вирішенням проблеми є виділення пам'яті за допомогою функції malloc().

5. СТРУКТУРИ

Структури у мові С є засобом доступу до записів або груп даних, причому кожний запис складається з полів. Синтаксис структур такий:

```
struct ім'я_структури
 {
 тип_1 поле_1;
 тип_2 поле_2;
 ……..
 тип_n поле_n;
};
```

Типовою помилкою є відсутність символу «;» після правої фігурної дужки.

Опис змінної типу struct:

```
struct struct_name x, y, z;
```

До полів структури можна звертатися, вказуючи ім'я змінної та ім'я поля і розділяючи ці імена крапкою:

```
type_1 val1=x. field_1;
```

Якщо задано вказівник ptr на структуру, то посилання на елемент структури матиме вигляд:

```
struct struct_name *ptr; type_1 val1=plr-> field 1;
```

Можна задати повне ім'я структури у вигляді макросу:

```
#define complex struct compl
float real;
float imag;};
```

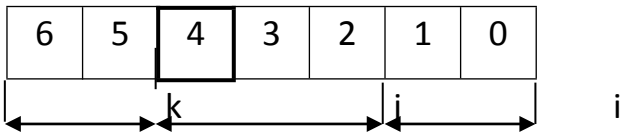
Після цього до елементів структури можна звертатися так: complex c1, c2;

```
c1.real= c2. Real / 3.14;
c2.imag=c1. imag * 2.0;
```

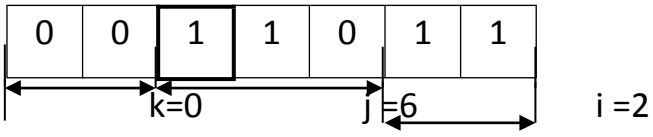
Бітові поля часто використовують в апаратно залежних програмних модулях. Деякі компілятори дозволяють вживати для бітових полів тільки тиа unsigned, інші – як int, так і unsigned int:

```
struct my_bits
{
        int i:2;
unsigned j:3;
int k:2;
} bit_field
bit_field. i=3; bit_field. j=6; bit_field. k=0;
```

Розподіл пам'яті для bit_field:



Бітове подання змінної bit_field:



Ініціалізація структур. Якщо структуру описано глобально (тобто поза функцією) або ж як static (усередині функції), то її можна одразу ініціалізувати в місці опису, наприклад:

```
#define REC struct data_rec
REC
```

```
{
char * last_name;
char * first_name;
long id_munber;
);
```

Далі структуру можна ініціалізувати таким чином:

```
REC my_rec = { "Bob", "Smith", 12345 };.
```

До елементів структури звертаються так:

```
char * str = my_rec. first_name;
long num = my_rec. id_number;
```

Об'єднання дозволяють в одній області пам'яті розміщувати дані різних типів. Об’єм виділеної під об'єднання пам'яті визначається довжиною найбільшого поля. Для опису об'єднання можна використати макрос:

```
#define data_un union data_rec
Data_un
{
char s[4];
int I;
float r;
};
Data_un dat;
int j=sizeof(dat);
j=4(max(s,I,r));
```

За допомогою об’єднань можна перетворювати типи даних. Для цього одному полю об’єднання присвоюють деяке значення, яке потім читається з

іншого поля (воно знаходиться в тій самій області пам’яті, що й перше), наприклад:

```
dat.i =5;
float fit= dat.r;                    /*fit=5.0*/
```

Оскільки ініціалізація всього об’єднання не має сенсу, виконується, наприклад така послідовність операторів:

```
dat.i =5;
dat.r=3.14;
```

У пам’яті залишається тільки останнє значення, тобто 3.14. Такий спосіб перетворення може виявитися немобільним.

6. ФУНКЦІЇ

Функції – це логічні елементи (блоки операторів), призначені для виконання дій, спрямованих на вирішення поставленого завдання. Для функцій введено поняття опису і визначення функції. Опис дає програмістові можливість одержати доступ до функції, а визначення – задавати дії, які вона має виконувати. До визначення функції входять:

1) тип даних, які функція вертає;
2) кількість і тип формальних параметрів;
3) код (тіло) функції (тобто набір дій функції);
4) інформація про видимість функції;
5) локальні змінні;

Якщо опис функції не містить її типу, то за замовчанням такій функції надається тип int. Значенням, яке вертається, не може бути масив, але може бути вказівник на масив.

Параметри у функцію передаються за значенням або за посиланням. Під час передачі параметрів за значенням всередині функції створюється копія параметра, наприклад:

void test_func (int first, Int second, Int*third)

У цьому прикладі значення, що вертається, має тип void, а функція має три параметри, два з яких передаються за значенням (створюються їх копії), третій – за посиланням (до функції передається адреса змінної third).

Локальні (автоматичні) змінні описують всередині функції. Межа видимості таких змінних - від точки опису локальної змінної до кінця блоку або функції, де цю змінну описано. Доступ до локальної із зовнішнього блоку неможливий. Пам'ять під таку змінну виділяється динамічно під час входження в блок і звільняється під час виходу з блоку.

Зовнішні змінні описують поза функцією і називають глобальними. Область видимості цих змінних – від точки опису змінної до кінця файлу. Пам'ять не вивільняється до закінчення роботи програми. Якщо оператора ініціалізації немає, то таким глобальним змінним автоматично надаються нульові значення.

Статичні змінні (дkя їх опису використовують специфікатор static) теж мають видимість від точки опису до кінця файлу. Ініціалізуються вони так само як і глобальні. Статичні змінні, описані всередині функції, зберігають свої значення при повторних викликах функції. На відміну від глобальних змінних до статичних змінних немає доступу з інших файлів проекту.

Рекурсія – це можливість викликати функцію із самої себе. При цьому великі обсяги обчислень можна записати декількома рядками програмного коду. Приклад рекурсії:

```
#include <studio. h>
main() {
extern void out_str(char * str, int cnt);          /*оголошення функції*/
out_str(“Take it easy”, 10);                       /*виклик функції*/
printf(“\n”);                                      /*щоб перейшов курсор*/
}
void out_str(char * str, int cnt)  {               /*визначення функції*/

if (cnt>0)    {
 printf(“\n%S”, str);
 out_str(str, cnt- -);    } }
```

Специфікатор extern свідчить, що функцію визначено поза функцією main(). Прикладом використання рекурсій є функції пошуку елементів, що входять у зв’язні списки деревоподібної структури.

7. БАЗОВІ ДИРЕКТИВИ ПРЕПРОЦЕСОРА

Препроцесор опрацьовує вихідний текст програми перед етапом компіляції, розширює всі макровиклики (макроси) і підставляє в текст програми зовнішні файли. У командах препроцесора використовують зарезервований символ #.

Деякі основні команди препроцесора:

#define – опис макросу;

#undef – скасування області дії макросу;

#include – підстановка тексту із зовнішнього файлу;

#ifdef – умовна підстановка залежно від визначення макросу;

#else – альтернатива для команди #ifdef;

#endif – кінець тексту, який підставляється умовно.

Умовну компіляцію можна застосувати для видалення тих частин програм, якими користуються тільки для налагодження програми, наприклад:

```
#define DEBUG
- код програми –
#ifdef DEBUG
- налагоджувальний код –
#endif
- основний код –
#ifdef DEBUG
- налагоджувальний код –
#else
- основний код –
#endif
- основний код –
```

Для перенесення частини команди препроцесора на новий рядок використовують символ «\». Командою #define препроцесору вказують на те, що всі входження ідентифікатора, який іде за сполученням символів #define, потрібно замінити відповідним блоком тексту. Команда #define може мати параметри, наприклад; #define sum(a,b,c) (a+b+c) .

Дозволено ставити пробіли між іменем макросу і лівою дужкою, а також відокремлювати пробілами елементи списку аргументів.

Приклад побічного ефекту команди #define: #define sq(x) x*x/. Припустимо, що цей макрос викликано так: int result = sq(j+2);
Якщо j=3, то очікуваний результат має дорівнювати 25. Насправді буде обчислено j+2*j+2, тобто результатом буде 11. Вийти із цієї ситуації можна, обравши коректну форму опису макросу, а саме #define sq(x) ((x)*(x)).

8. ПОМИЛКИ ПРОГРАМУВАННЯ, ТЕСТУВАННЯ ПРОГРАМ

Існує багато розповсюджених помилок, які допускають програмісти, коли вперше починають працювати з мовою С. Далі розглядаються деякі поширені помилки та надаються рекомендації щодо їх уникнення.

Імена шляхів у рядках С. Загальновідомо, що в MS-DOS зворотний слэш (\) позначає ім'я директорії (каталогу). Але у рядках С зворотний слеш є escape-символом. Тому часто виникають проблеми при заданні імені шляху за допомогою рядка С.

Наприклад, в операторі

```
file = fopen("c:\new\too1s.dat", "r");
```

Можна було б припустити, що відкривається файл TOOLS.DAT у директорії NEW накопичувача С:. Однак це не вірно. В цьому випадку \n – це escape-послідовність для символу нового рядка (LF), а \t — символ табуляції.

В результаті, на початку та в кінці ім‘я файлу міститиме символи нового рядка й табуляції відповідно. Тому DOS розглядає такий рядок як неправильне ім'я файлу. Правильним є наступний рядок:

```
"c:\\new\\tools.dat"
```

Правильне та неправильне використання вказівників. Для програмістів, які починають працювати на С, вказівники є джерелом найбільшої кількості помилок, пов'язаних з використанням оператора непрямої адресації (*) та оператора адреси (&).

Застосування неініціалізованого вказівника. Основна серйозна помилка полягає в присвоєнні значення адресі, що зберігається у вказівнику, без призначення адреси власне вказівникові. Наприклад,

```
main ()
{
        int *iptr;
        *iptr = 421;
        printf("*iptr = %d\n", *iptr);
}
```

Ця помилка є надзвичайно небезпечною, тому що вказівник iptr містить деяку довільну адресу, за якою запам'ятовується значення 421. У прикладі програма досить мала, тому пошкодження інформації в пам'яті комп'ютера малоймовірно. Однак у більших програмах така ймовірність зростає, адже цілком можливо, що за адресою, на яку випадково вказує iptr, зберігається інша інформація. При використанні моделі пам'яті tiny, в якій сегменти коду та даних займають спільну область пам'яті, існує ймовірність пошкодження власне машинного коду. Рекомендується виділяти пам'ять за допомогою malloc та присвоювати адресу виділеної пам'яті вказівнику.

Рядки. Відомо, що рядки можна оголошувати як вказівники на char або як масиви char. Між двома засобами оголошення є одна важлива відмінність: при використанні вказівника на тип char пам'ять для рядка не виділяється, а при використанні масиву виділяється пам'ять, причому змінна масиву містить адресу пам'яті.

Нерозуміння розглянутої відмінності може викликати помилки двох типів, як продемонстровано у наступній програмі:

```
main()
{
```

```
        char *name;
        char msg[10];
        printf("Ваше ім'я? ");
        scanf("%s", name);
        msg ="'Привіт, ";
        printf('"%s%s", msg, name);
}
```

На перший погляд програма виглядає абсолютно правильною. Але тут припущено дві різні помилки. Перша помилка пов'язана з оператором

```
scanf("%s", name)
```

Власне оператор цілком коректний: name — вказівник на char, перед даним ім’ям не потрібен оператор адреси (&).

Однак програма не виділяла пам'ять для пате; у результаті введене ім'я буде збережено за довільною адресою. При компіляції видається попередження Possible use of 'name' before definition ("можливе використання 'name' раніше його визначення"). Але це не розглядається компілятором як помилка.

Друга проблема, пов'язана з оператором msg = "Привіт, ", обов'язково викличе помилку. Компілятор вважатиме, що виконується спроба заміни msg значенням адреси рядкової константи "Привіт, ". Це неприпустимо, адже імена масивів є константами і їх модифікація не дозволяється. (Так, якщо 7 — константа, оператор "7=i" неправильний). Компілятор видасть повідомлення про помилку Lvalue required ("Необхідне L-Значення").

Найпростіший спосіб усунення подібних помилок заснований на інвертуванні способів оголошення пате та msg:

```
main()      {
```

```
        char name;
        char *msg[10];
        printf("Ваше ім'я? ");
        scanf("%s", name);
        msg = "Привіт, ";
        printf("%s%s", msg, name);
}
```

Ця програма працює правильно. Для змінної name виділено область пам'яті, що дозволяє зберігати введене ім'я, вказівнику msg присвоюється адреса рядкової константи "Привіт, ".

Проте якщо необхідно залишити попередні оголошення, програму слід змінити:

```
main()      {
        char *name;
        char msg[10];
        name = (char *) malloc(10);
        printf("Ваше ім'я? ");
        scanf("%s", name);
        strcpy(msg, "Привіт, ");
        printf("%s%s", msg, name);
}
```

Виклик malloc() виділяє 10 байт пам'яті та присвоює змінній пате адресу пам'яті, що усуває першу проблему. Функція strcpy() виконує посимвольне копіювання строкової константи "Привіт, " у масив msg.

Змішування, присвоєння (=) і рівність (= =). У мовах Паскаль і Бейсік перевірка рівності виконується за допомогою виразу типу if (a = b). У С така конструкція також припустима, однак має зовсім інший зміст.

Розглянемо наступний фрагмент програми:

```
If (a = b)
        puts("Рівно");
else
        puts("Не рівно");
```

Програмісти, що працюють на мовах Паскаль або Бейсік, можуть припустити, що повідомлення Рівне буде видано в тому випадку, коли a і b мають однакові значення, та повідомлення Не рівне — в іншому випадку. Проте відбувається зовсім інше. У С вираз a = b означає, що "змінній a слід присвоїти значення b", причому весь вираз приймає значення b. Тому у попередньому фрагменті після присвоєння значення b змінній a буде видане повідомлення Рівне, якщо значення b ненульове, а якщо ні, то повідомлення Не рівне.

Насправді потрібно наступне:

```
If (a == b)
        puts{"Рівно");
else
        puts("Не рівно");
```

Відсутність break в операторах switch. Слід пам'ятати, що в операторі switch для завершення певного варіанта використовується оператор break. Якщо не вказати оператор break для деякого варіанта, то виконаються всі наступні варіанти.

Індексація масивів. Не слід забувати, що масиви починаються з [0], а не з [1]. Одна із загальних помилок представлена в наступному прикладі:

```
main()
{
        int list[l00], i;

        for (i = 1; i <= 100; i++)
                list[i] = i*i;
}
```

У цій програмі першому елементу list, а саме list[0], значення не присвоюється, причому одне зі значень запам'ятовується в неіснуючому елементі list[100], що, можливо, призведе до пошкодження інших даних.

Правильним є наступний код:

```
main()
{
        int list[l00], i;

        for (i = 0; i < 100; i++)
                list[i] = i*i;
}
```

Неможливість передачі за адресою. У наведеній нижче програмі є помилка:

```
main()
{
        int a, b, sum;
        printf("Введіть два значення: ");
        scanf('%d %d", a, b);
```

```
        sum = a + b;
        printf("Сума рівна %d\n", sum);
    }
```

Помилка присутня в операторі scanf("%d %d", a, b), оскільки замість значень в scanf слід передавати адреси. Це справедливо для всіх функцій, формальними параметрами яких є вказівники. Попередня програма може бути скомпільована й виконана, оскільки scanf буде розглядати довільні значення, передані в a і b, як адреси для запам'ятовування значень, що вводяться.

Правильним є оператор scanf("%d %d", &a, &b); тут в scanf передаються адреси a і b, тому введені значення правильно запам'ятовуються в даних змінних. Аналогічні помилки можуть виникати і у функціях, визначених користувачем, наприклад у функції swap, визначеній в розділі, присвяченому вказівникам. Розглянемо наступний виклик swap:

```
main()      {
        int i,j;
        i = 421;
        j = 53;
        printf(" До: i = %4d j = %4d\n", i, j);
        swap(i,j);
        printf("Після: i = %4d j = %4d\n", i,j); }
```

До та після виклику swap значення змінних i та j однакові, але значення з адресами даних 421 і 53 будуть переставлені, що може створити проблеми, які досить складно локалізувати.

Для уникнення таких помилок рекомендується використовувати прототипи функцій та повні визначення функцій.

Якщо використовувати визначення функції swap, наведене раніше в цьому розділі, тоді при компіляції main компілятор видасть повідомлення про помилку.

Однак програма буде нормально компілюватися, якщо визначити swap у такий спосіб:

```
void swap(a, b)
int *a, *b;    {
        ...      }
```

Винесення визначень a та b за дужки блокує перевірку помилок, що є найкращим доводом на користь відмови від класичного стилю визначення функцій.

9. ФАЙЛИ

За допомогою файла дані, що обробляються програмою, можуть бути отримані ззовні, а результати програми передані у зовнішній світ.

Розмір фізичного файла обмежується лише ємністю диска (носія).

Фізичний файл – послідовність байтів в зовнішній пам'яті.

Байт	Байт	Байт	…	…	…	…	…	…	Константа EOF (-1)

Логічний (внутрішній) файл – той, що описаний і використовується в програмі. Логічний файл описується в програмі і представляє собою вказівник на початок фізичного файла. В програмі файл описується:

FILE *<ім'я файла>;

Наприклад:

#include <stdio.h>

FILE *f, *g;

Структура файла представляє собою послідовність байтів; в мові С файли неструктуровані і нетипізовані.

Розрізняють 2 види файлів:

- бінарні;
- текстові (хоча за своєю структурою всі файли є бінарними).

Бібліотека С підтримує три рівня вводу-виводу:

- потоковий ввід-вивід;
- ввід-вивід нижнього рівня;
- ввід-вивід для консолі портів (залежить від конкретної ОС)

На рівні потокового вводу-виводу обмін даними виконується побайтно, тобто за одне звернення до пристрою (файла) виконується зчитування або запис фіксованої порції даних (512 або 1024 байта). При вводі з диску або при зчитуванні з файла дані розміщуються в буфері ОС, а потім побайтно або

порціями передаються програмі. При виводі в файл дані так само накопичуються в буфері, а при заповненні буфера записуються у вигляді єдиного блока на диск. Буфери реалізуються у вигляді ділянок основної пам'яті.

Таким чином, ***потік*** – це файл разом з наданими засобами буферизації.

Операції над файлами. Оскільки при описанні файла в програмі указується вказівник на нього, то операції, які припустимі до файлів – це вказівникові операції:

- операція отримання адреси (&),
- розадресація (розіменування) або опосередковане звернення до об'єкта(*),
- присвоювання, додавання з константою, віднімання, арифметичні операції з вказівниками: інкремент (++), декремент (--), порівняння, приведення типів.

Функції бібліотеки С, що підтримують обмін даними на рівні потока, дозволяють обробляти дані різних розмірів і форматів. При роботі з потоком можна:

1. Відкривати і закривати потоки (при цьому вказівники на потік зв'язуються з конкретними файлами).
2. Вводити і виводити дані або їх порції встановленої довжини.
3. Керувати буферизацією потока і розміром буфера.
4. Отримувати і встановлювати вказівник поточної позиції (поточна – доступна для читання-запису) в файлі.

1. Відкриття файла.

Перед початком роботи з потоком його потрібно відкрити. При цьому потік зв'язується зі структурою передвизначеного типа FILE, визначення якої знаходиться в <stdio.h>. В структурі знаходиться вказівник на буфер, вказівник на поточну позицію, і т.п.

FILE *fopen(const char *path, const char *mode);

const char*path – рядок з іменем файла, зв'язаного з потоком,

const char*mode – рядок режима відкриття файла

Наприклад:

FILE *f=fopen("C:\\t.txt","r");//t.txt – ім'я файла, r – режим відкриття файла.

Режими відкриття файла:

r	Відкрити існуючий файл для читання, вказівник на початку файла
w	Відкрити файл для запису. Якщо файл непустий, то вміст губиться, якщо файл відсутній, то він створюється
a	Відкрити файл для дозапису. Якщо файл існує, то дозапис виконується в кінець
t	Відкрити файл як текстовий
b	Відкрити файл як бінарний
+	Дозволити і читання, і запис

Потік може бути відкритий в:

- текстовому (t) режимі (потік розглядається як сукупність рядків, в кінці кожного є керуючий символ '\n') (наприклад, fopen ("t.txt", "rt") означає, що текстовий файл t.txt відкритий для читання. В різних ОС кінець рядка може кодуватися різними символами). ***За замовчанням встановлюється текстовий режим.***
- двійковому (b) режимі (потік розглядається як набір двійкової інформації).

Помилки, які можуть виникати при відкритті потока:

- файл, що зв'язаний з потоком, не знайдений (при читанні з файла);
- диск заповнений (при записі);
- диск захищений від запису.

В усіх цих випадках вказівник набуває значення NULL (0).

2. Закриття файла.

Функція fclose( ) закриває файл.

int fclose(FILE *fp);

Повертає нуль у випадку успіху і EOF у випадку невдачі.

Інформація реально записується повністю в файл тільки в момент його закриття, до цього вона може міститися в оперативній пам'яті (в файловій кеш-пам'яті), що при виконання багаторазовних операцій читання-запису значно прискорює роботу програми.

При нормальному завершенні програми функція викликається автоматично для кожного відкритого файла.

3. Бінарне читання

Дані функції називаються бінарними тому, що не виконують ніякого перетвореня інформації при її вводі або виводі (невелике виключення при роботі з текстовими файлами, яке ми розглянемо пізніше): інформація зберігається у файлі як послідовність байтів рівно в тому вигляді, як вона зберігається в пам'яті комп'ютера.

Після того, як файл відкритий відповідним чином, можна читати інформацію в нього, або записувати. Для читання використовується функція

int fread(void*ptr,int size, int n, FILE*f), де

void*ptr – вказівник на область пам'яті, в якій розміщуються зчитані з файла дані,

int size – розмір одного елемента, що зчитується,

int n – кількість елементів, що зчитуються,

FILE*f – вказівник на файл, з якого виконується зчитування.

У випадку успішного зчитування функція повертає кількість зчитаних елементів, інакше - EOF.

4. Бінарний запис

Для блокового бінарного запису використовується функція

int fwrite(void*ptr,int size, int n, FILE*f), де

void*ptr – вказівник на область пам’яті, в якій розміщуються дані для запису,

int size – розмір одного елемента, що записується,

int n – кількість елементів, що записуються,

FILE*f – вказівник на файл, в який виконується запис.

У випадку успішного запису функція повертає кількість записаних елементів, інакше - EOF.

5. Кінець файла

В <stdio.h> визначена константа EOF, яка позначає кінець файла (від’ємне ціле число -1). Функція

int feof(FILE*f)

повертає ненульове значення, якщо в результаті виконання останньої операції читання з потоку досягнуто кінця файла.

6. Встановлення вказівника на задану позицію

int fseek(FILE *f, long off, int org);

Функція повертає 0, якщо зміщення виконано успішно, інакше повертає ненульове значення (наприклад, якщо вказане зміщення некоректне при заданій операції, або файл чи потік не дозволяє прямий доступ)

FILE *f – вказівник на файл,

long off - позиція зміщення,

int org – початок відліку (звідки відраховувати зміщення).

Зміщення (в байтах) задається виразом або змінною і може бути від’ємним, тобто переміщення може бути в обох напрямках.

Початок відліку задається однією з визначених в <stdio.h> констант:

SEEK_CUR	Зміщення відраховується від поточної позиції
SEEK_SET	Зміщення відраховується від початку файла
SEEK_END	Зміщення відраховується від кінця файла

7. Отримання кількості байт відносно початку потоку

long ftell(FILE *f);

Повертає -1 у випадку невдачі, номер поточного байта у випадку удачного виконання.

8. Встановити значення вказівника на початок потоку

void rewind(FILE *f);

9. Форматне введення

int fscanf(FILE*fp, const char* Формат, СписокАдр),

де FILE*f – вказівник на файл, з якого відбувається читання,

const char*fmt – форматний рядок,

СписокАдр – список адрес змінних, в які заноситься інформація з файла.

- Виконує форматоване (аналогічно scanf()) ***читання*** значень змінних з файла, зв'язаного з файлом, вказаним як перший параметр.
- Функція повертає число змінних, яким присвоєне значення.
- Файл, зв'язаний з потоком, мусить бути відкритий як текстовий, в режимі, що допускає читання (див. fopen()). Заголовочний файл: <stdio.h>

Функція fscanf() читає інформацію з текстового файла, де інформація записана в звичному для людини вигляді, і перетворює її у внутрішнє представлення даних в пам'яті комп'ютера. Дані при вводі-виводі перетворюються в їх текстове представлення відповідно до форматного рядка.

10. Форматне виведення

int fprintf(FILE*fp, Формат, СписокЗмінних),

де FILE*f – вказівник на файл, в який виконується запис,

const char*fmt – форматний рядок,

СписокЗмінних - список змінних, що записуються в файл.

- Виконує форматований ввід (аналогічно printf()) в файл, зв'язаний з потоком, указаним як перший параметр.
- Повертає число записаних символів.

  - Файл, зв'язаний з потоком, мусить бути відкритий як текстовий, в режимі, що допускає запис (див. fopen()). Заголовочный файл: <stdio.h>

11. Введення символа з потоку f.

int fgetc(FILE *f);

Повертає код введеного символа або константу EOF у випадку кінця файла або помилки читання.

12. Виведення символа в потік f.

int fputc(int c, FILE *f);

Повертає константу EOF при помилці, при успішному зчитуванні - код виведеного символа c (невід'ємне значення).

13. Зчитування рядка з потока

char *fgets(char *line, int size, FILE *f);

- зчитує рядок з потоку f і записує його в масив символів line розміру size. Максимальна довжина зчитаного рядка на одиницю менша, ніж size, оскільки завжди в кінець зчитаного рядка додається нульовий байт.
- функція сканує вхідний потік до тих пір, поки не зустріне символ перевода рядка '\n' або поки число введених символів не стане

рівним size-1. Символ перевода рядка ‘\n’ також записується в масив перед нульовим байтом.

- повертає вказівник line в випадку успішного виконання або нульовий вказівник при помилці або в кінці файла.

14. Виведення рядка в потік

char *fputs(char *line, FILE *f);

- вивести рядок line в потік f. Нуль в кінці рядка не записується.
- У випадку успіха повертає невід’ємне значення, у випадку помилки EOF.

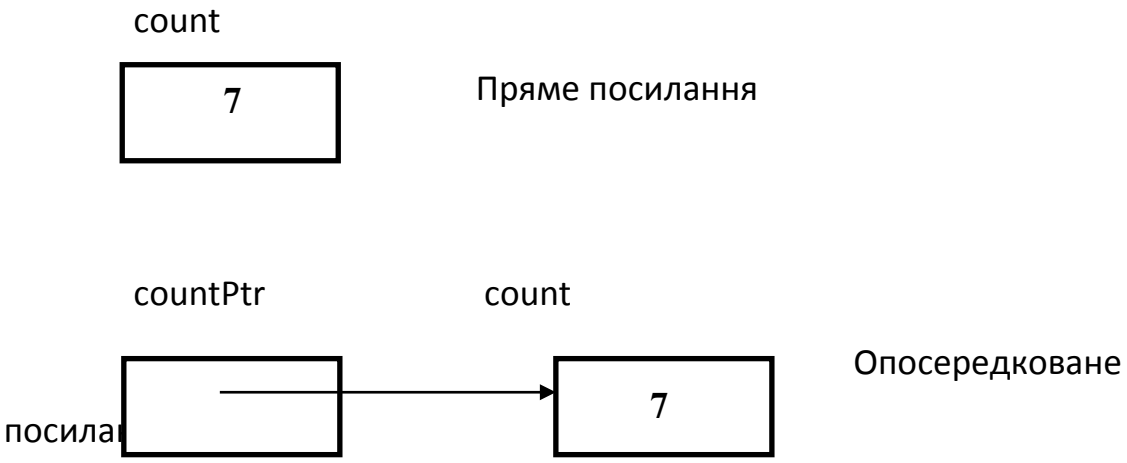
10. ВКАЗІВНИКИ

Вказівники – проста і логічна концепція, але така, що вимагає уваги до дрібниць. Правильне використання вказівників має велике значення з наступних причин:

1. Вказівники надають спосіб, який дозволяє функціям модифікувати аргументи, що передаються.
2. Вказівники використовуються для підтримки системи динамічного розподілу пам’яті.
3. Використання вказівників може підвищити ефективність роботи деяких підпрограм.
4. Вказівники, як правило, використовуютьсчя для підтримки деяких структур даних, таких, як зв’язані списки і двійкові дерева.

Оголошення і ініціалізація вказівників

Звичайна змінна містить визначене значення. Змінні-вказівники містять адреси пам’яті. Якщо ім’я змінної відсилає до значення прямо, то вказівник – опосередковано. Програміст може визначити власні змінні для збереження адрес областей пам’яті, і це будуть вказівники.



Вказівники, як і інші змінні, перед використанням мусять бути оголошені. Для цього використовується знак *:

int *countPtr;

char *point;

int *countPtr, point; // помилка, оскільки вказівником є лише countPtr.

Вказівники можна оголошувати, щоб вказувати на об’єкти будь-якого типа даних, крім посилання і бітового поля. Вказівник не є самостійним типом, він завжди зв’язаний з певним іншим типом.

Існують дві категорії вказівників, які відрізняються властивостями і набором операцій:

- вказівник на об’єкт певного типа (в тому числі і на тип void)
- вказівник на функцію

Вказівник на void використовується:

- коли конкретний тип об’єкта, адресу якого потрібно зберігати, не визначений (наприклад, якщо в одній і тій же змінній в різні моменти часу потрібно зберігати адреси об’єктів різних типів).

Вказівнику на void можна:

- присвоїти значення вказівника будь-якого типа
- порівнювати його з будь-якими вказівниками, але перед будь-якими діями з адресою пам’яті, на яку він посилається, потрібно привести його до конкретного типа явним чином.

Наприклад:

```
int i;
void *p;
p = &i;// можно присвоїти вказівнику p адресу змінної i, але змінити значення вказівника не можна
char * message; // вказівник на char
int *array[10]; // масив з 10 вказівників на int
```

```
int (*pointer)[10];/*вказівник pointer на масив з 10 елементів типу int*/
struct list{
int *count;
struct  list *next;} line;/* структура, кожне з полів якої є вказівником*/
struct list *next;// вказівник на структуру list
int **Ptr; //вказівник на вказівник
```

Вказівник може бути константою або змінною, а також вказувати на константу або змінну.

Наприклад:

```
int i; // ціла змінна
const int ci = 1; // ціла константа
int *pi ; // вказівник на цілу змінну
const int *pci; // вказівник на цілу константу
int *const cp = &i; // вказівник-константа на цілу змінну
const int *const cpc = &ci; // вказівник-константа на цілу константу
```

Вказівник-константа на неконстантні дані - вказівник, що завжди указує на одну і ту саму комірку пам’яті, де дані можна міняти з допомогою вказівника. Ім’я масива – константний вказівник. Розмір вказівника 4 байти.

Після оголошення вказівника і до першого присвоювання йому значення, він може містити невідоме значення. Якщо використати неініціалізований вказівник, це може привести до критичної помилки.

Ініціалізуються вказівники:

- при оголошенні,
- в операторі присвоєння.

Вказівник може бути в таких станах:

1. невизначеним (int *a,*b;);

2. визначеним (містити 0 або адресу):

1.1. 0 або NULL – вказівник ні на що не вказує. Константа NULL визначена в stdio.h.

2.2. адреса існуючого об’єкта:

- через операцію отримання адреси

```
int a=5;
int *p=&a; // або int p(&a);
```

- з допомогою іншого проініціалізованого вказівника

```
int *r=p;
```

- адреса присвоюється в явному вигляді

```
char*cp=(char*)0x B800 0000;
```

де 0x B800 0000 – шістнадяцяткова константа, (char*) – операція приведення типа.

- з допомогою імені масива або функції, які трактуються як адреса (див. «Масиви», «Передача імен функцій як параметрів):

```
int b[10]; // масив
int *t = b; // присвоєння адреси початку масива
void f (int a){/* ... */} // визначення функції
void (* pf) (int);// вказівник на функцію
pf = f; // присвоювання адреси функції
```

Операції над вказівниками

З вказівниками можна виконувати наступні дії:

1) операція отримання адреси (&),

2) розадресація (розіменування) або опосередковане звернення до об’єкта (*),

3) присвоювання,

4) додавання з константою,

5) віднімання константи,

6) інкремент (++),

7) декремент (--),

} арифметичні операції з вказівниками

8) порівняння,

9) приведення типів.

1) & (операція адресації) , взяття адреси.

Унарна операція, яка розміщує адресу свого операнда в указаний вказівник. Операція отримання адреси не може бути використана для отримання адреси скалярного виразу, неіменованої константи або регістрової змінної (клас пам'яті register).

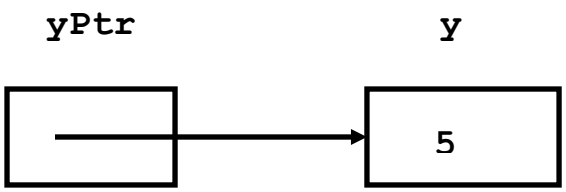
Наприклад, якщо

int y = 5;

int *yPtr;

то оператор

yPtr = &y;



присвоює адресу змінної y вказівнику yPtr, тобто змінна yPtr вказує на y.

2) * (операція опосередкованої адресації або розіменування)

Повертає значення об'єкта, на який вказує її операнд (тобто вказівник).

Наприклад, оператор

printf("%d",*yPtr);

виведе значення змінної y, а саме 5. Використання * подібним чином називається *розіменуванням вказівника.*

Розіменований вказівник може використовуватися в лівій частині оператора присвоювання, наприклад *yPtr = 9.

Розіменування вказівника, який не був ініціалізований або якому не присвоєна ніяка адреса, може викликати помилку виконання програми. Також помилкою буде спроба розіменувати невказівник.

3) Присвоювання вказівників

Як і звичайні змінні, вказівники можуть використовуватися справа від оператора присвоювання для присвоювання значення іншому вказівнику.

```
main (){
int x;
int *p1, *p2;
p1 = &x;
p2 = p1;
printf ("%p %p", p1, p2); }/*виводить адреси p1 і p2, вони однакові,
містять адресу x*/
```

4) Арифметичні операції

Застосовуються лише до вказівників одного типа. Вони мають смисл в основному при роботі зі структурами даних, що послідовно розташовані в пам’яті, наприклад, з масивами.

- Інкремент /декремент переміщує вказівник на величину sizeof (тип).

- Різниця двох вказівників це різниця їх значень, поділена на розмір типа в байтах.

- Додавання вказівника і константи. Додавання двох вказівників не допускається.

- Порівняння вказівників. Можливо порівняти два вказівники (==, !=, <, >).

- Приведення типів

При записі виразів з вказівниками потрібно звертати увагу на пріоритет операцій.

Вказівник на масив

При визначенні масива йому виділяється пам'ять. Після цього ім'я масива сприймається як константний вказівник того типа, до якого відносяться елементи масива.

Мова Сі інтерпретує ім'я масива як адресу його першого елемента.

Операції над іменем масива (вказівник на перший елемент):

1) Операція sizeof() дає розмір масива.

2) Результатом операції & до масива є адреса нульового елемента масива:

ім'я масива == &ім'я масива == &ім'я масива[0],

3) до імені масива можна застосовувати всі правила адресної арифметики.

ім'я масива[індекс] - це вираз з двома операндами: ім'я масива і індекс.

Ім'я масива – це вказівник-константа, а індекс визначає зміщення від початку масива.

З використанням вказівників, звернення за індексом можна записати таким чином:

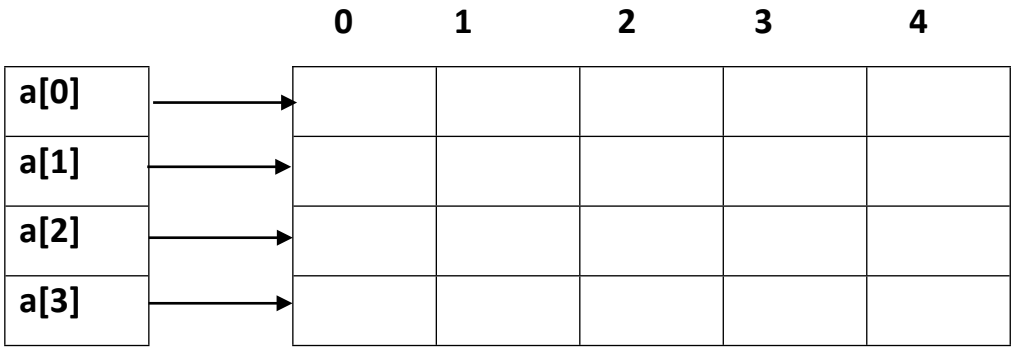
*(ім'я масива + індекс)

Тобто, a[3]=1 еквівалентно *(a+3)=1.

Вказівники на багатовимірні масиви і масиви вказівників

Оскільки значення багатовимірного масива зберігаються в пам'яті послідовно, є припустимим звертання вигляду *p++.

Але багатовимірний масив можна трактувати і як масив, елементи якого теж є масивами. Наприклад, масив з описанням int a[4][5] можна подати як масив з 4 вказівників типа int*, які містять адреси одномірних масивів з 5 цілих елементів.



При такому способі для доступу до елемента масива можна використовувати вказівник на вказівник, тоді наприклад *(*(a+1)+1) буде означати звернення до елемента a[1][1].

Приклад доступу по вказівниках до елементів багатовимірного масива:

```
#include <stdio.h>
int main (){
int b[3][2][4] = {  0,   1,   2,   3,
                   10,  11,  12,  13,
                  100, 101, 102, 103,
                  110, 111, 112, 113,
                  200, 201, 202, 203,
                  210, 211, 212, 213        };
//адреса масива b[][][]
printf( "\nb=%x ", b);
//адреса масива b[0][][]
```

```
printf ( "\n*b=%x ", *b);
//адреса масива **b[0][0][]
printf ( "\n **b=%x",  **b);
//Елемент b[0][0][0]
printf ( "\n***b=%x", ***b)
//адреса масива b[1][][]
printf ( "\n*(b+1)=%x", *(b+1));
//адреса масива b[2][][]
printf ( "\n*(b+2)=%x" ,*(b+2));
//адреса масива b[0][1][]
printf ( "\n*(*b+1)=%x", *(*b+1));
printf ( "\n*(*(*(b+1)+1)+1)=%d", *(*(*(b+1)+1)+1));
printf ("\n*((b[1][1]+1)=%d",*(b[1][1]+1));
//елемент b[2][0][0]
printf ("\n*((b[1]+1)[1]=%d",*(b[1]+1)[1]);
return 0;}
```

Якщо потрібно передати масив вказівників у функцію, можна використати спосіб, аналогічний передачі звичайних масивів, тобто надо викликати функцію з іменем масива без індексів.

Функція, що отримує масив x:

```
void display_array(int *q[]){//q –масив вказівників на цілі
int t;
for(t=0; t<10; t++)
printf ("%d ", *q[t]);
}
```

Проблеми, пов'язані з вказівниками

Вказівники є потужним засобом і потрібні в більшості програм. Але їх невірне використання приводить до критичних помилок.

Якщо використовувати неправильний вказівник, то наприклад, при читанні може бути прочитане «сміття», а при записі затерти ділянки кода або данних.

Не існує очевидного способа для розв'язання проблем, пов'язаних з вказівниками. Краще не допускати цих проблем.

Розглянемо дві найпоширеніші помилки.

1. НЕініціалізований вказівник

```
// програма невірна
int main (void) {
int x, *p;
x = 10;
*p = x;
return 0; }
```

Помилка в тому, що значення 10 присвоюється деякій невідомій ділянці пам'яті. Вказівник не отримав ніякої адреси, він невизначений.

2. Некоректне використання вказівника

```
#include <stdio.h>
// програма невірна
int main(void){
int x, *p;
x = 10;
p = x;
```

```
printf ("%d", *p);
return 0; }
```

Помилка - у невірному операторі присвоювання p = x. Потрібно виправити p = &x;

Вказівники на вказівники

Може бути описаний вказівник на вказівник, наприклад

```
float **nb;
```

Для отримання значення, на яке вказує вказівник на вказівник, потрібно двічі записати оператор *:

```
#include <stdio.h>
int main(void){
int x, *p, **q;
x = 10;
p = &x;
q = &p;
printf ("%d", **q) ; //вивід значення x=10
return 0;}
```

Вказівники на структури

Вказівники на структури визначаються так, як і вказівники на інші типи. Наприклад:

student*ps;

Можна ввести вказівник для типа, який не має імені:

```
struct{
char *name;
int age; } *person;//вказівник на структуру
```

При визначенні вказівник на структуру може бути одразу проініціалізований:

```
student *ps=&mas[0];
```

Вказівник на структуру забезпечує доступ до її елементів 2 способами:

1.(*вказівник).ім'я_елемента printf("%s",(*ps).name);

2. вказівник->ім.'я_елемента printf ("%s", ps->title);

Вказівник на функцію

Вказівник на функцію містить адресу, за якою розташований виконуваний код функції, тобто адресу, по якій передається керування при виклику функції.

Вказівники на функцію використовуються для опосередкованого виклика функції (не через її ім'я, а через звернення до змінної, що зберігає її адресу), а також для передачі імені функції в іншу функцію як параметр.

```
тип (*ім'я) ( список_типів_аргументів );
```

Наприклад, оголошення int (*fun) (double, double); задає вказівник з іменем fun на функцію, що повертає значення типа int і має два аргументи типа double.

У визначенні вказівника кількість і тип параметрів мусять співпадати з відповідними типами в визначенні функції, на яку ставиться вказівник.

Виклик функції з допомогою вказівника має вигляд:

(*ім’я_вказівника)(список фактичних параметрів);

Наприклад:

```
void f1()
{printf("\nfunction f1");}
void f2()
{printf("\nfunction f2");}
main(){
void(*ptr)(); // вказівник ptr на функцію
ptr=f2; // вказінику ptr присвоюється адреса функції f2
(*ptr)(); // виклик функції f2
ptr=f1; // вказвнику ptr присвоюється адреса функції f1
(*ptr)(); // виклик функції f1 з допомогою вказівника
}
```

Якщо ім’я функції використовувати без наступних дужок і параметрів, то воно буде виступати як вказівник на цю функцію, і його значенням буде адреса розташування функції в пам’яті. Це значення можна буде присвоїти іншому вказівнику.Тоді цей новий вказівник можна буде використовувати для виклику функції.

1. При визначенні вказівник на функцію може бути одразу проініціалізований.
2. Вказівники на функції доцільно використовувати в тих випадках, коли функцію треба передати в іншу функцію як параметр.
3. Щоб зробити програму легко читаємою, при описанні вказівників на функції використовують перейменування типів typedef.
4. Вказівники на функції передаються в підпрограму таким же чином, як і параметри інших типів

11. ЕЛЕМЕНТИ ООП

Будь-яка програма, написана мовою ООП, відображає в своїх даних стан фізичних предметів або абстрактних понять – об'єктів програмування, для роботи з якими вона призначена.

ООП дозволяє розкласти проблему на складові частини, кожна з яких стає самостійним об'єктом. Кожний з об'єктів містить свій власний код і дані, що належать цьому об'єкту.

Всі дані про об'єкт програмування і його зв'язки з іншими об'єктами можна об'єднати в одну структуровану змінну. В першому наближенні її можна назвати об'єктом.

З об'єктом зв'язаний набір дій, інакше називаних методами. Фактично це функції, які отримують як обов'язковий параметр вказівник на об'єкт і виконують певні дії з даними об'єкта програмування. Технологія ООП забороняє працювати з об'єктом інакше, чим через методи, і таким чином, внутрішня структура прихована від зовнішнього користувача.

Об'єкт – це структурована змінна, що містить всю інформацію про певний фізичний предмет або про поняття, що реалізується в програмі.

Клас – це описання множини об'єктів програмування і виконуваних над ними дій (тобто об'єктний тип).

В С і в інших процедурно-орієнтованих мовах програмування прагне бути орієнтованим на дії, тоді як в ідеалі програмування на С++ об'єктно-орієнтоване: в С одиницею програмування є функція, в С++ - клас, на основі якого створюються об'єкти.

Отже, кожний клас містить дані і набір функцій, що маніпулюють з ними. Компоненти-дані класа називаються *data members* або ***даними***. Компоненти-функції, які визначають дії над даними класа, називаються *member functions* або ***методами***.

Класи в С++ є природнім продовженням структури struct в С.

Конкретні величини типа даних «клас» називаються ***екземплярами*** класа, або ***об'єктами*** (об'єкт-змінна).

Об'єкти взаємодіють між собою, посилаючи і отримуючи повідомлення. ***Повідомлення*** - це запит на виконання дії, що містить набір необхідних параметрів. Механізм повідомлень реалізується через виклик відповідних функцій.

Тобто з допомогою ООП реалізється так звана «подійно-керована модель», коли дані активні і керують викликом того або іншого фрагмента програмного кода (наприклад, будь-яка програма, керована через меню).

В найпростішому випадку клас можна визначити таким чином:

тип_класа ім'я_класа {список_членів_класа}; де

тип_класа – одне із службових слів class, struct, union;

ім'я_класа – ідентіфікатор;

список_членів_класа – визначення і описання типізованих даних і функцій, що належать класу.

Наприклад:

```
struct date  {          // дата
int month,day,year;       // поля: місяць, день, рік
void set(int,int,int);    // метод – встановити дату
void get(int*,int*,int*);  // метод – отримати дату
void next();      // метод – встановити наступну дату
```

```
void print();           // метод – вивести дату
};

class complex           // комплексне число
{
double re,im;
double real(){return(re);}
double imag(){return(im);}
void set(double x,double y){re = x; im = y;}
void print(){cout<<“re = “<<re; cout<<“im = “<<im;}
};
```

Для описання екземпляра використовується конструкція ім’я_класа ім’я_об’єкта;

Наприклад:

```
date today, my_birthday;
date *point = &today;   // вказівник на об’єкт типа date
date clim[30];          // масив об’єктів
date &name = my_birthday; // посилання на об’єкт
```

Звертатися до даних об’єкта і викликати функції для об’єкта можна двома способами.

1. Перший – за допомогою кваліфікованих імен:

ім’я_об’єкта.ім’я_даного

ім’я_об’єкта.ім’я_функції

Наприклад:

```
complex x1,x2;
```

```
x1.re = 1.24;
x1.im = 2.3;
x2.set(5.1,1.7);
x1.print();
```

2. Другий спосіб доступа використовує вказівник на об’єкт:

вказівник_на_об’єкт–>ім’я_компонента

Наприклад:

```
complex *point = &x1; // або point = new complex;
point –>re = 1.24;
point –>im = 2.3;
point –>print();
```

Поля класа:

1) можуть мати будь-який тип, крім типа цього ж класа (але можуть бути вказівниками і посиланнями на цей клас);

2) можуть бути описані з модифікатором const, при цьому вони ініціалізуються лише один раз (з допомогою конструктора) і не можуть змінюватися;

3) можуть бути описані з модифікатором static, але не як auto, extern і register;

4) ініціалізація полів при описанні не допускається.

В класі відображені найважливіші концепції ООП: інкапсуляція, наслідування, поліморфізм.

Інкапсуляція даних – механізм, який об’єднує дані і код, що маніпулює з цими даними, а також захищає і те, і інше від зовнішнього втручання або неправильного використання. В ООП код і дані можуть бути об’єднані разом (в так званий «чорний ящик») при створенні об’єкта.

Всередині об’єкта коди і дані можуть бути відкритими або закритими.

Закриті коди або дані доступні лише для інших частин того ж самого об’єкта, і відповідно недоступні для тих частин програми, які існуюьб поза об’єктом.

Відкриті коди і дані, навпаки, доступні для всіх частин прогрми, в тому числі і для інших частин того ж самого об’єкта.

Наслідування. Новий, або похідний клас може бути визначений на основі вме існуючого, або базового класа.

При цьому новий клас зберігає всі властивості старого: дані об’єкта базового класа включаються в дані об’єкта похідного, а методи базового класа можуть бути викликані для об’єкта похідного класса, причому вони будуть виконуватися над даними включеного в нього об’єкта базового класа.

Інакше кажучи, новй клас наслідує як дані старого класа, так і методи їх обробки.

Якщо об’єкт наслідує свої властивості від одного предка, то це одиночне наслідування. Якщо об’єкт наслідує дані і методи від декількох базових класів, то це множинне наслідування.

Поліморфізм – це властивість, яка дозволяє один і той же ідентифікатор (ім.’я) використовувати для розв’язку двох і більше схожих, але технічно різних задач.

Метою поліморфізма в ООП є використання одного імені для завдання дій, спільних для ряда класів об’єктів. Такий полиморфізм ґрунтується на можливості включення в дані об’єкта також і інформації про методи їх обробки (в вигляді вказівників на функції).

Будучи доступним в певній точці програми, об’єкт, навіть при відсутності повної інформації про його тип, завжди може коректно викликати властиві йому методи.

Концепція структури/класа в C++ (на відміну від C) дозволяє членам структури бути загальними, приватними або захищеними:

- public – спільні;
- private – приватні;
- protected – захищені. Використання protected зв'язано з наслідуванням.

Наприклад:

class <ім'я>{

[private:]

<описання прихованих елементів>

public:

<описання доступних елементів>

}; //

- Елементи, що описані після private, видимі лише всередині класа. Цей вид доступа прийнятий за замовчанням.
- Загальнодоступні (public) компоненти доступні в будь-якій частині програми, вони можуть використовуватися будь-якою функцією як всередині класа, так і поза ним. Доступ ззовні виконується через ім'я об'єкта:

ім'я_об'єкта.ім'я_члена_класа

ссилка_на_об'єкт.ім'я_члена_класа

вказівник_на_об'єкт->ім'я_члена_класа

- Захищені (protected) компоненти доступні всередині класа і в похідних класах.

- Дія будь-якого специфікатора розповсюджується до наступного специфікатора або до кінця класа.

- Можна задавати декілька секцій private і public, порядок їх слідування значення не має.

- Стандартним є розміщення даних в приватній частині (private), а частини методів – в загальній частині (public). В цьому випадку private визначає дані об’єкта і службові функції, а методи загальної частини реалізують методи роботи з об’єктом.

Наприклад:

```
class complex {
double re, im;          // private за замовчанням
public:
double real(){return re;}
double imag(){return im;}
void set(double x,double y){re = x; im = y;}
};
```

Кожний клас має окрему область видимості.

a. Імена членів класа локальні в цьому класі.
b. Область видимості даних і методів не залежить від точки їх оголошення, вони є видимими у всьому класі.
c. Кожний метод визначає свою власну локальну область видимості, подібно звичайній функції.

Можна явно указати область видимості в С++, використовуючи оператор разрешения контекста (області видимості) :: .

Він має найвищий пріоритет, і має дві форми:

- унарну – посилається на зовнішній контекст . Використовується для звернення до імені, що відноситься до зовнішнього контексту

і що приховане локальним контекстом або контекстом класа, має вигляд

:: Идентификатор

- бінарну – посилається на контекст класа

ИмяКласса :: Идентификатор

Наприклад:

```
#include <iostream>
using namespace std;
int count = 0; // (*)
void func(void) {
  for (int count = 0; count<10; count++)   {
    ++ ::count;  // збільшує на 1 (*)
  } }
int main() {
  cout << "count =" << ::count << endl;
  func();
  cout << "count =" << ::count << endl;
  cin.get();
  return                                    0;                                    }
```

Бінарна форма використовується для посилання на контекст класа з метою видалення неоднозначності імен, які можуть повторно використовуватися всередині класа.

```
class cl1 {void f() {…} …};
class cl2 {void f() {…} …};
cl1 :: f();  // звернення до f() з cl1
cl2 :: f();  // звернення до f() з cl2
```

Класи можуть бути вкладеними.

Наприклад:

```
#include <iostream>
using namespace std;
int c;  // зовнішня область (контекст)
class X  // зовнішнє оголошення класа
{
public:
  static int c;
  class Y  // внутрішнє оголошення класа
  {
  public:
    static int c;
    static void f(int key)
    {
      ::c = key;      // зовнішня змінна
      X::c = key + 1; // змінна з класа X
      c = key + 2;    // змінна з класа Y
    }
  };
};
int X::c = 0;
int X::Y::c = 0;
int main(){
  X::Y::f(3);
  cout << "   c = " << c << endl;
  cout << "X::c = " << X::c << endl;
```

```
 cout << "Y::c = " << X::Y::c << endl;
 cin.get();
 return 0; }
```

1) C++ дає можливість створювати вкладені методи з використанням вкладених класів. Це обмежена форма вкладеності функції. Методи мусять визначатися всередині локального класа, і на них не можна посилатися всередині цього контекста.
2) Як і в С, звичайні вкладені функції заборонені. Вкладений клас не знаходиться в області дії включаючого його класа.
3) Якщо всередині класа записаний лише прототип метода, сам метод мусить бути визначений в іншому місці програми з допомогою операції доступа до області видимості ::.

Наприклад:

```
class Tree {
        int siz;
        char area;
   public:
Tree(int s = 10, char a = 'a'){ siz = s; area = a;}//прототип
        int _up_ (int d, int y, bool light, bool wet);
        int _size_ (){return siz;}
        int _area_(){return area;}
        };// Tree

        int Tree::_up_(int d, int y, bool light, bool wet)
        {/* тіло метода */
        if (light && wet) y+=d;
        return y;}
```

Класи можуть бути глобальними (оголошеними поза будь-яким блоком) і локальними (оголошеними всередині блока, наприклад, функції або іншого класа).

Зауваження по локальному класу:

• всередині локального класа можна використовувати типи, статичні (static) і зовнішні (extern) змінні, зовнішні функції і елементи перелічення з області,Ю в якій він описаний; забороняэться використовувати запрещается автоматичні змінні з цієї області;

• локальний клас не може мати статичні елементи;

• методи цього класа можуть бути описані лише всередині класа;

• якщо один клас вкладений в інший, то вони не мають яких-небудь особливих прав доступа до елементів один одного і можуть звертатися до них тільки за загальними правилами.

➢ Метод можна визначити як вбудований поза класом з допомогою директиви inline (як і для звичайних функцій, вона носить рекомендаційний характер):

```
inline int Tree::_area()
{
return area;
}
```

➢ Можна створити константний метод, значення полів якого змінювати заборонено. До нього мусять застосовуватися лише константі методи:

```
Class Tree{
int _area_() const {return area;}
};
const Tree Chestnut(20,’n’); // Константний об’єкт
```

cout<<Chestnut._area_();

Зауваження по константному методу:

• оголошується з ключовим словом const після списку параметрів;

• не може змінювати значення полів класа;

• може викликати лише константі методи;

• може викликатися для любих (не лише константних) об’єктів;

• рекомендується описувати как константні ті методи, що призначені для отримання значення полів.

Конструктор

В кожному класі є хоча б один метод, ім’я якого співпадає з іменем класса. Він називається конструктором, призначений для ініціалізації об’єкта і викликається автоматично при створенні об’єкта класа.

Коли клас має конструктор, всі об’єкти цього класа будуть ініціалізуватися.

Формат визначення конструктора:

ім’я_класа(список_формальних_параметрів)

{оператори_тіла_конструктора}

Наприклад: розглянемо клас date:

```
class date  {
  int day, month, year;
public:
  date(int, int, int); // конструктор
};
```

Конструктор виділяє пам’ять для об’єкта і ініціалізує дані – елементи класа.

Зауваження:

• Для конструктора не визначається тип значення, що повертається, навіть void не припустимий.

• Вказівник на конструктор не може бути визначений, тому відповідно не можна отримати адресу конструктора.

• Конструктори не наслідуються.

• Конструктори не можуть бути описані з ключовими словами: virtual, static, const, mutuable, volatile.

• Клас може мати декілька конструкторів з різними параметрами для різних видів ініціалізації (працює механізм перевантаження).

• Конструктор, що викликається без параметрів, називається конструктором за замовчанням.

• *Параметри конструктора* можуть мати будь-який тип, крім цього ж класа (але може посилання на нього (T&)). Можна задавати значення параметрів за замовчанням, їх може мати лише один із конструкторів.

• При спробі створення об’єкта класа, що містить константи або посилання, буде видана помилка, оскільки їх необхідно ініціалізувати конкретними значенням, а конструктор за замовчанням цього робити не вміє.

• Конструктори глобальних об’єктів викликаються до виклику функції main. Локальні об’єкти створюються, як тільки стає активною область їх дії. Конструктор запускається і при створенні тимчасового об’єкта (наприклад, при передачі об’єкта з функції).

• Якщо не заданий жодний конструктор, компілятор створює його автоматично для будь-якого класа. За замовчанням створюється конструктор без параметрів і конструктор копіювання.

• За замовчанням конструктори створюються загальнодоступними (public).

Якщо потрібно забезпечити декілька способів ініціалізації об’єктів класа, то можна задати декілька конструкторів:

```
class date {
 int month, day, year;
public:
 date(int, int, int); // день месяц год
 date(char*); // дата в строковом представлении
 date(); // дата по умолчанию: сегодня
};
```

Конструктори підлягають тим же правилам щодо типsв параметрів, що і перевантажені функції: якщо конструктори різняться по типах своїх парметрів, то компілятор при кожному використанні може обрати правильний:

```
date july4(«Февраль 27, 2014»);
date guy(27, 2, 2014);
date now; // инициализируется по умолчанию
```

Одним із способів зменшити число перевантажених функцій (в тому числі і конструкторів) є використання значень за замовчанням.

Конструктор за замовчанням

Конструктор, що не потребує параметрів, називається контструктором за замовчанням. Це може бути конструктор з пустим списком параметрів або конструктор, в якому всі аргументи мають значення за замовчанням.

Конструктори можуть бути перевантажені, але конструктор за замовчанням може бути лише один.

```
class date {
  int month, day, year;
public:
```

```
 date(int, int, int);
 date(char*);
 date(); // конструктор за замовчанням
};
```

При створенні об’єкта викликається конструктор, за виключенням випадку, коли об’єкт створюється як копія іншого об’єкта цього ж класа, наприклад:

```
date date2 = date1;
```

Однак є випадки, коли створення об’єкта без виклика конструктора здійснюється неявно:

- формальный параметр – об’єкт, що передається по значенню, створюється в стеку в момент виклика функції і ініціалізується копією фактичного параметра;
- результат функції – об’єкт, що передається по значенню, в момент викорнання оператора return копіюється в тимчасовий об’єкт, що зберігає результат функції.

В усіх цих випадках транслятор не викликає конструктор для нового створюваного об’єкта:

- date2 в наведеному визначенні;
- створюваного в стеку формального параметра;
- тимчасового об’єкта, що зберігає значення, що повертається функцією.

Замість цього в них копіюється вміст об’єкта-джерела:

- date1 в наведеному прикладі;
- фактичного параметра;
- об’єкта-результата в операторі return.

Конструктор копії (копіювання)

Конструктор копіювання обов’язковий, якщо в програмі використовуються функції-елементи і перевизначені операції, які отримують

формальні параметри і повертають як результат такий об’єкт не по посиланню, а по значенню.

При наявності в об’єкті вказівників на динамічні змінні і масиви або ідентифікаторів зв’язаних ресурсів, таке копіювання вимагає дублювання цих змінних або ресурсів в об’єкті-приймачі. З цією метою вводиться конструктор копіювання, який автоматично викликається в усіх перерахованих випадках. Він має єдиний параметр-посилання на об’єкт-джерело:

```
class String {
  char *str;
  int size;
public:
  String(String&); // Конструктор копіювання
};
String::String(String& right) { // Створює копії динамічних //змінних та ресурсів
  s = new char[right->size];
  strcpy(str,right->str);
}
```

Конструктор копіювання викликається що разу, коли виконується копіювання об’єктів, що належать класу.

Він викликається:

а) коли об’єкт передається в функцію по значенню;

б) при побудові тимчасовго об’єкта як значення, що повертається функцією;

в) при використанні об’єкта для ініціалізації іншого об’єкта.

Існує два способи ініціалізації даних об’єкта черех конструктор.

Перший – передача значень параметрів в тіло конструктора (див.вище).

Другий – застосування списка ініціалізаторів даного класа. Цей список поміщається між списком параметрів і тілом конструктора. Кожний ініціалізатор списка відноситься до конкретного компонента і має вигляд:

ім’я_данного (вираз)

Наприклад:

```
class CLASS_A
{
 int i;
 float e;
 char c;
public:
CLASS_A(int ii,float ee,char cc) : i(8), e( i * ee + ii ),c(cc){}
   ...
   };
```

Клас “символьний рядок”:

```
#include <string.h>
#include <iostream.h>
class string
{
char *ch; // Вказівник на текстовий рядок
int len;    // довжина текстового рядка
public:
// конструктори:
// створення об’єкта – пустого рядка
string(int N = 80): len(0){ch = new char[N+1]; ch[0] = ‘\0’;}
// створює об’єкт по заданому рядку
string(const char *arch){
```

```
    len = strlen(arch);
     ch = new char[len+1];
     strcpy(ch,arch);
}
      // методи
      // повертає посиалння на довжину рядка
int & len_str(void){return len;}
      // повертає вказівник на рядок
      char *str(void){return ch;}
//    . . .
};
```

Деструктор

Визначений користувачем клас має конструктор, який забезпечує належну ініціалізацію.

Деструктор — це особливий вид метода для вивільнення пам'яті, що займається об'єктом. Деструктор викликається автоматично, коли об'єкт виходить з області видимості:

• для *локальних* об'єктів - при виході з блока, в якому вони оголошені;

• для *глобальних* - як частина процедури виходу з main:

• для *об'єктів, заданих через вказівники^* деструктор викликається неявно при

використанні операції delete.

При виході з області дії вказівника на об'єкт автоматичний виклик деструктора не виконується.

Формат деструктора:

```
~імя_класа( ) { оператори_тіла_деструктора}
```

```
class date {
  int *day, *month, *year;
public:
  date(int d, int m, int y) {
    day = new int;
    month = new int;
    year = new int;
    *day = d;
    *month = m;
    *year = y;
   }
~date() { delete day; delete month; delete year; }
};
```

Зауваження:

- Ім’я деструктора співпадає з іменем його класа, і перед ним ставиться “~” (тильда).
- Деструктор не має параметрів і значення, що повертається.
- Виклик деструктора виконується неявно (автоматично), як тільки об’єкт класа знищується. Наприклад, при виході за область визначення або при виклику оператора delete для вказівника на об’єкт:

```
string *p=new string (“строка”);
delete p;
```

- Якщо в класі деструктор не визначений явно, то компілятор генерує деструктор за замовчанням, який просто вивільняє пам’ять, зайняту даними об’єкта. У випадках, коли потрібно вивільнити і інші об’єкти пам’яті, наприклад, область, на яку вказує ch в объекте string, необхідно визначити деструктор явно:

~string(){delete []ch;}

- Як і для конструктора, не може бути визначений вказівник на деструктор.
- Не може бути оголошений як const або static.
- Не наслідується.
- Може бути віртуальним (див.«Виртуальные методы», Павл., с. 205).
- Описувати в класі деструктор явним чином потрібно, коли об’єкт містить вказівники на пам’ять, що виділена динамічно, інакше при знищенні об’єкта пам’ять, на яку посилались його поля-вказівники, не буде відміченя як вільна.

Деструктор можна викликати явним чином через указання повністю уточненого імені, наприклад

```
string *s; ...
s -> ~string();
```

Це може бути корисним для об’єктів, яким з допомогою перевантаженої операції new виділялась конкретна адреса пам’яті.

Можна явно не викликати деструктор без необхідності (не рекомендується).

Додаток 1. СПИСОК РЕКОМЕНДОВАНОЇ ЛІТЕРАТУРИ

1. *Берри Р., Микинз Б. Язык Си: Введение для программистов – М.: Финансы и статика, 1988. – 191 с.*
2. *Болски М. И. Язык программирования Си. – М.: Радио и связь, 1988. – 96 с.*
3. *Кернига́н Б., Ритчи Д. Язык программирования Си. М.: Вильямс. – 2013. – 304 с.*
4. *Кернига́н Б., Ритчи., Фьюэр А. Язык программирования Си: Задачи по языку Си. – М.: Финансы и статистика, 1985. – 279 с.*
5. *Паппас К., Мюррей У. Visual C++: Руководство для профессионалов.- СПб: BHV – Санкт-Петербург, 1996. -912 с.*
6. *Уинер Р. Язык Турбо Си. – М.: Мир, 1991. - 384 с.*
7. *Вирт, Н. Алгоритмы и структуры даних / Н. Вирт. - СПб.: Невский диалект, 2005. – 352 с.*
8. *Кнут, Д. Искусство программирования для ЭВМ / Д. Кнут. - М.: Наука, 1988. – 1200 с.*
9. *Трой Д. программирование на языке Си для персонального компьютера. М.Ж Радио и связь. – 1991. – 428 с.*
10. *Х.Дейтел, П.Дейтел. Как программировать на С. М.: Бином, 2017. – 1000с.*

Додаток 2. ІНТЕГРОВАНЕ СЕРЕДОВИЩЕ РОЗРОБКИ

Для виконанння лабораторних робіт може бути використане інтегроване середовище розробки (ICP) Code :: Blocks.

Перший запуск ICP Code::Blocks

Після запуска ICP Code::Blocks, під час її завантаження з’явиться заставка, на якій відображена його версія.

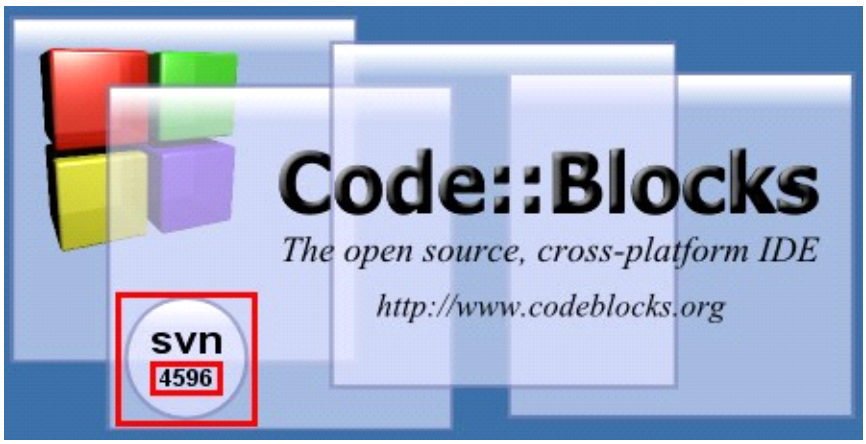


Рис. 1. Заставка Code::Blocks

Після цього з’явиться діалогове вікно зі списком знайдених компіляторів; з нього потрібно обрати той, що буде використаний за замовчанням. Для виконання практичних робіт потрібно вибрати пропонований за замовчанням **GNU GCC Compiler** і натиснути **OK**. На рис. 2 наведений цей діалог, де кольором виділений пропонований за замовчанням компілятор.

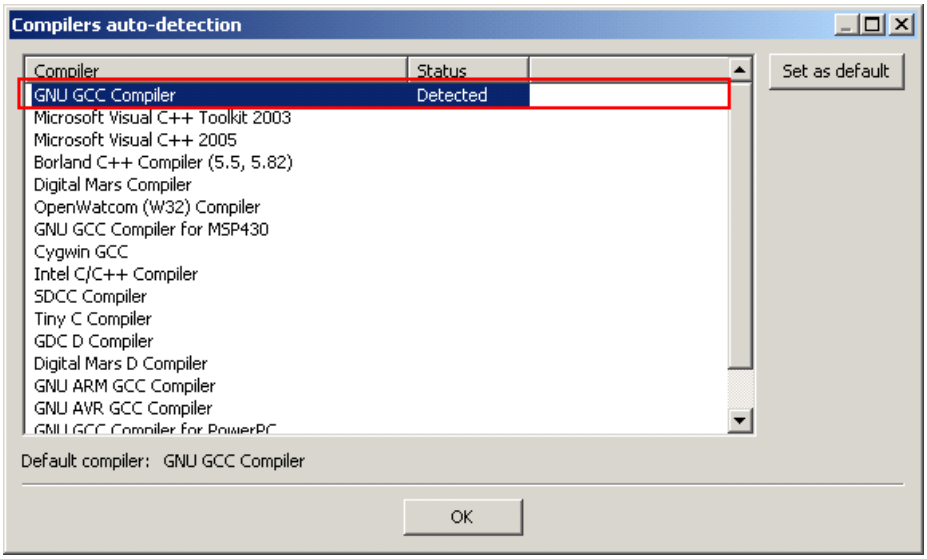


Рис. 2. Вибір компілятора за замовчанням

На рис. 3 представлене ділогове окно, яке з’являється при кожному запуску. Для того, щоб воно не з’являлось, треба зняти маркер «Show tips at startup».

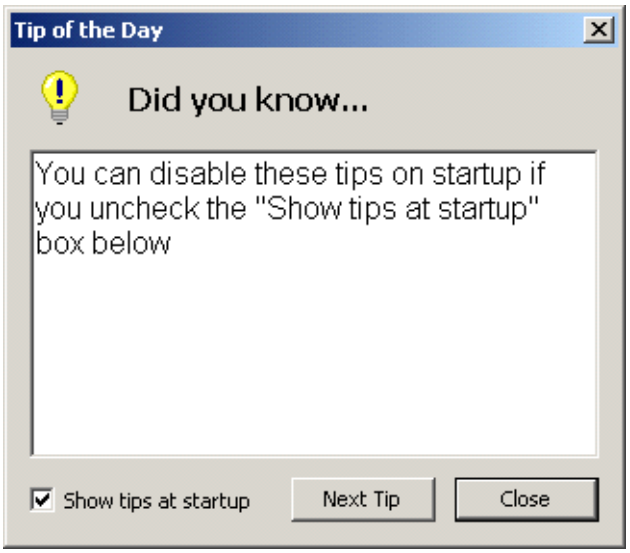


Рис. 3. Діалогове вікно

Далі ICP запропонує використати її як програму за замовчанням для всіх файлів .c та .cpp (див. рис. 4).

- «No, leave everything as it is» - відмова від цього.

- «No, leave everything as it is (but ask me again next time)» - відмова зараз, але питаннч буде повторене в подальших завантаженнях.
- «Yes, associate Code::Blocks with C/C++ files types» - цією програмаю будуть відкриватися лише файли C/C++
- «Yes, associate Code::Blocks with every supported type (including project files from other IDEs)» - всі файли, які підтримуються Code::Blocks, будуть ним відкриватися.

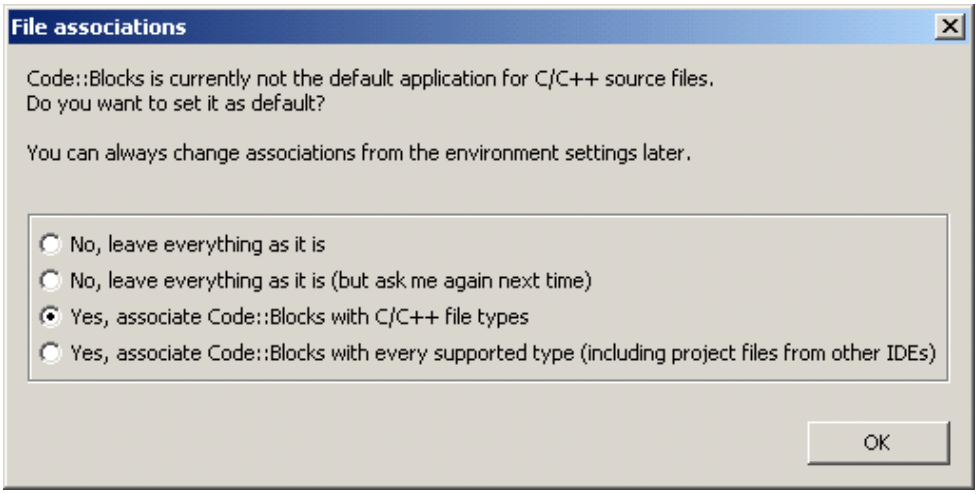


Рис. 4. Асоціація файлів з програмою Code::Blocks

Після натискання OK, відбудеться перехід до головного вікна програми (див. рис.5).

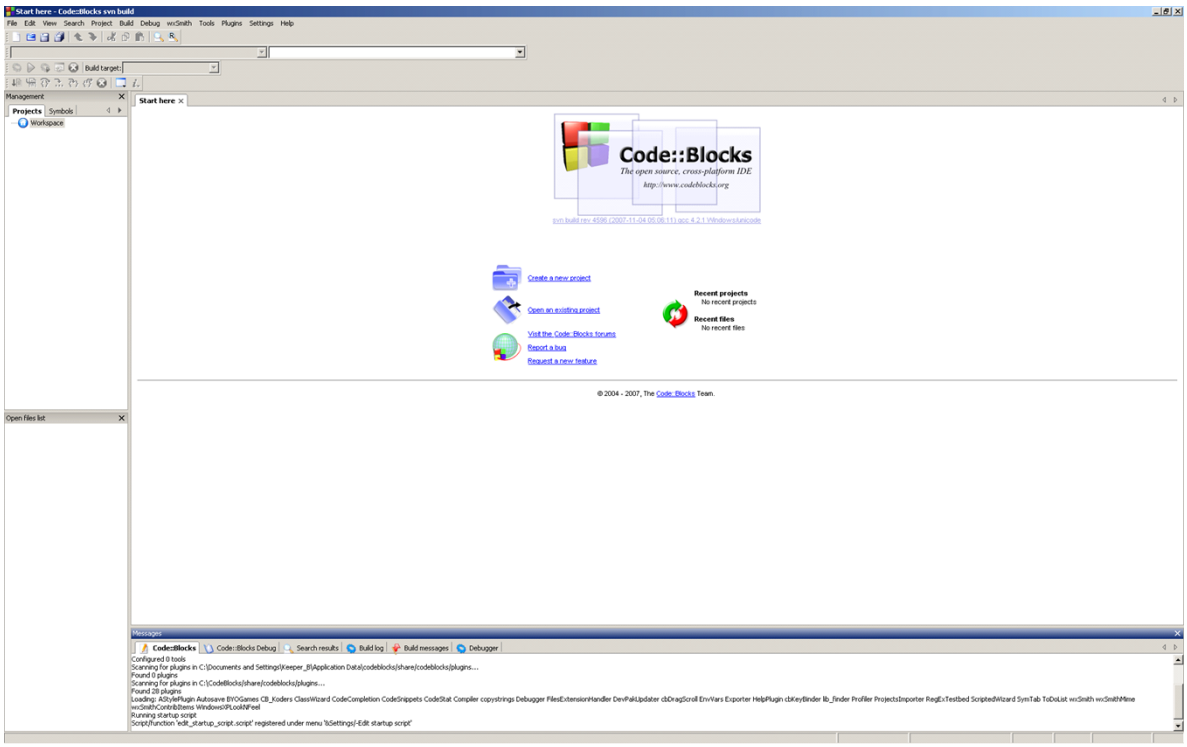


Рис. 5. Основне вікно ICP Code::Blocks

Створення проекта в Code::Blocks

Create a new project запускає майстер «створення проектів».

Open an existing project відкриває діалогове вікно **Open file**, в якому можна відкрити файл уже існуючого проекта. Діалогове вікно **Open file** можна викликати клавішами **Ctrl+O**.

Пункт меню **File->New->Project...** – створення нового проекта (або у вікні «Швидкого старту» пункт **Create a new project)**.

У діалоговому вікні **New from template**, що з'явилося, потрібно вказати шаблон проекта. Для виконання лабораторних робіт можна обрати **Console application** (див. рис. 6), натиснути **GO**, після чого з'явиться майстер створення консольних додатків.

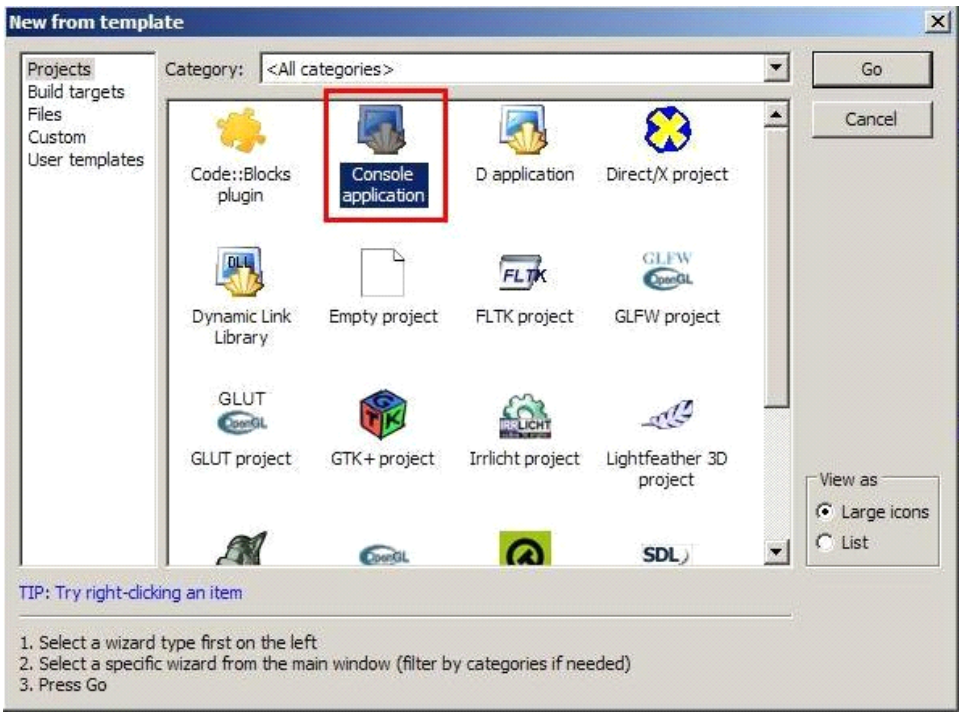


Рис. 6. Діалогове вікно вибору шаблона проекта New from template

Якщо в діалогове вікно майстра (див. рис.7) поставити галочку **Skip this page next time**, то при створенні нового проекта ця сторінка майстра буде пропускатися.

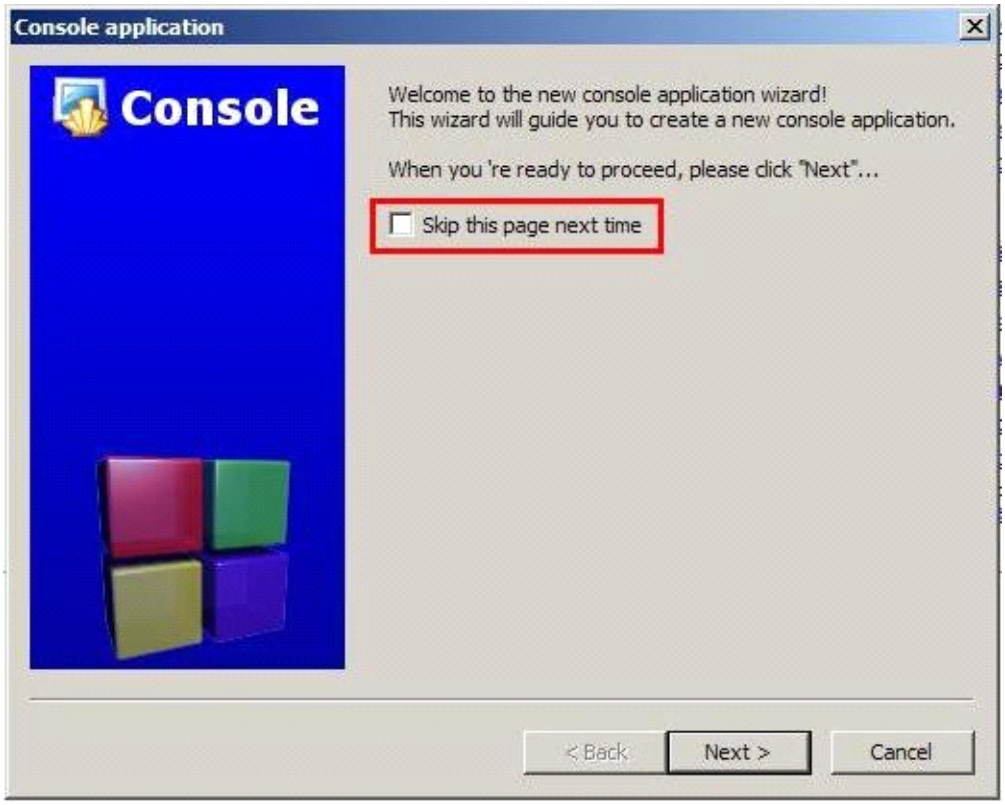


Рис. 7. Діалогове вікно майстра

В полі **Project title** (рис. 8) потрібно указати ім’я проекта, в полі **Folder to create project in** - папку, де він буде зберігатися.

Необов’язкові для заповнення поля **Project filename –** указати ім’я файла проекта, **Resulting filename** **-** вказується кінцеве ім.’я каталога і проекта.

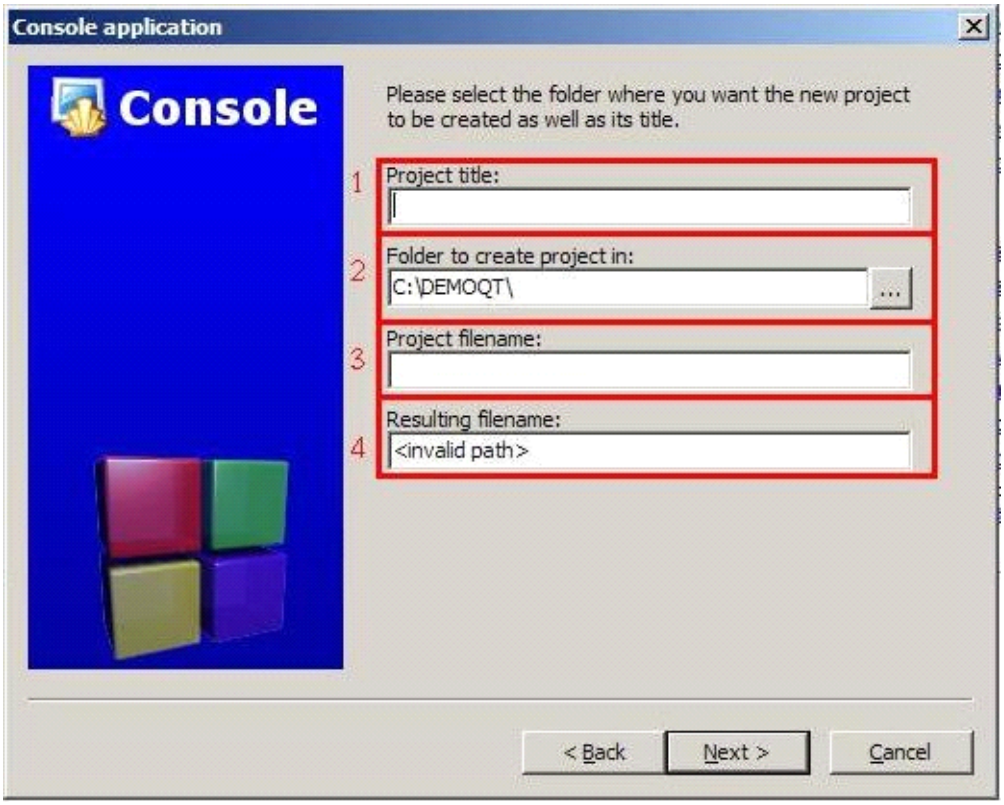


Рис. 8. Сторінка майстра для указання імені проекта і папки, де він буде збережений.

На сторінці (див. рис. 9) потрібно обрати компілятор, який буде використаний при компіляції додатку.

Сценарії зборки допомагають отримати декілька версій одного додатку:

- **Debug** – сценарій компіляції, використовується при від лагодженні додатку;
- **Release** – сценарій компіляції готового додатку.

Якщо немає необхідності створювати певний сценарій, то потрібно зняти відповідну позначку.

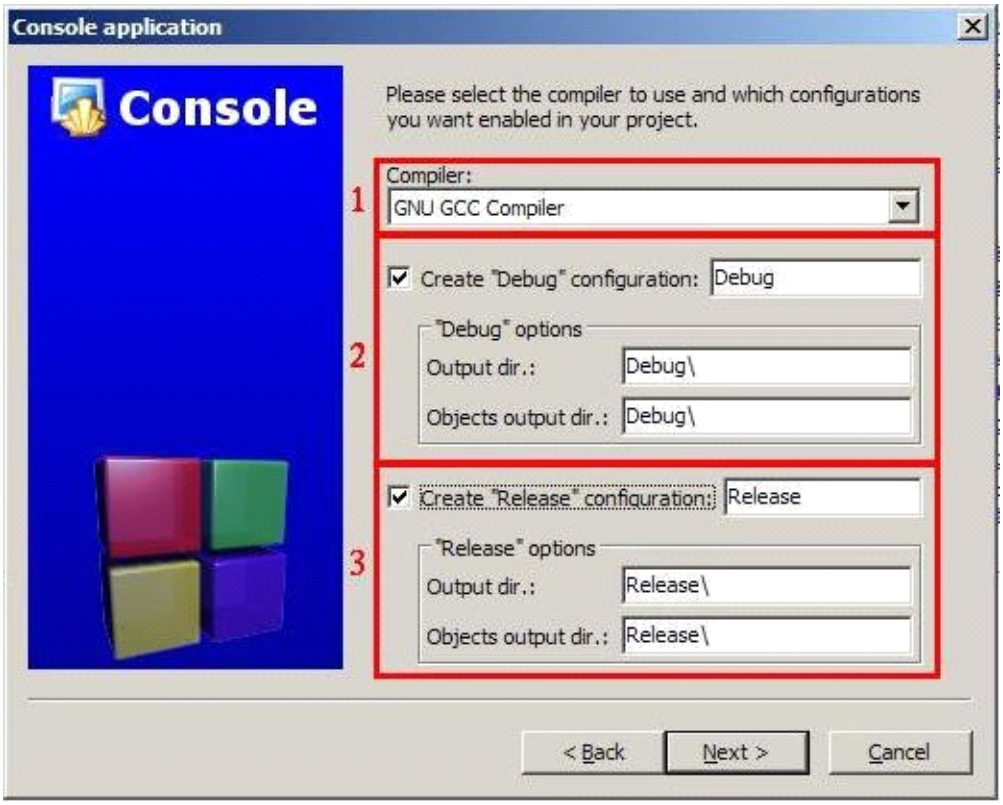


Рис. 9. Сторінка майстра для вибору компілятора і сценарія зборки

В кожному сценарії є можливість задати каталоги, куди будуть поміщатися файли скомпільованого додатку:

- Output dir – для всіх файлів;

- Objects output dir - для об’єктних файлів.

На фінальній сторінці (див. рис. 10) потрібно вибрати мову програмування.

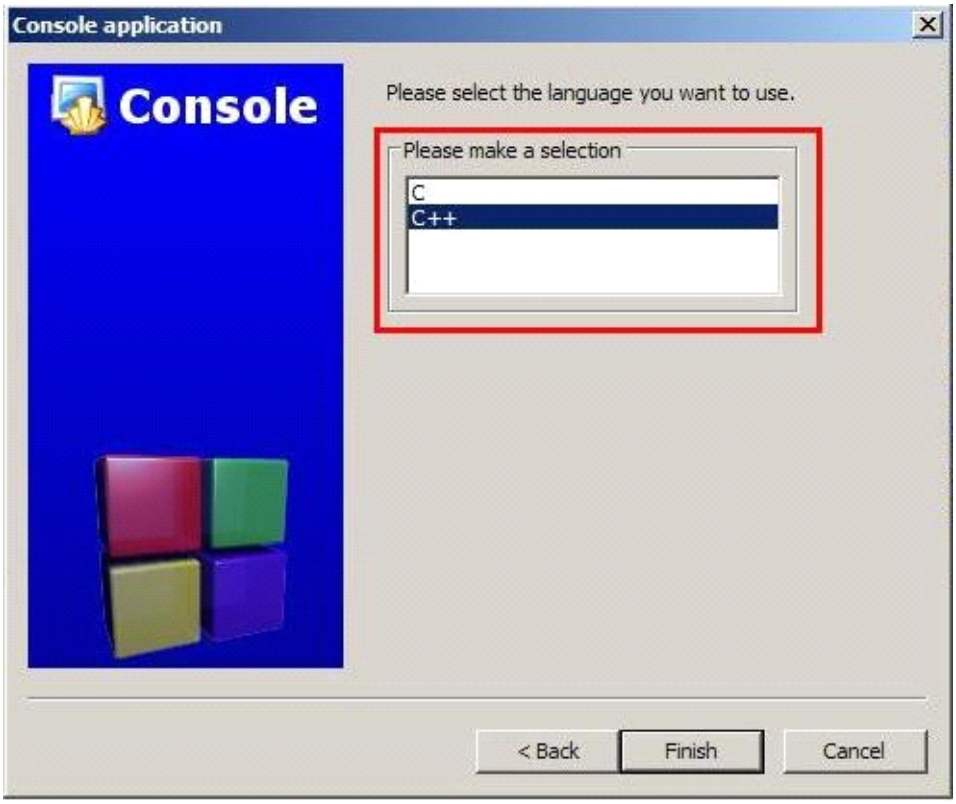


Рис. 10. Сторінка для обрання мови додатку

Після натискання на **Finish** проект буде створено і відкрито.

У вікні редактора кода зліва розташована панель **Management**, де у вигляді дерева відображена ієрархічна структура проекта, що складається з одного файла main.cpp, який знаходиться всередині віртуального каталога *Sources*. Каталог *Sources* знаходиться всередині створеного проекта. Проект належить робочому простору *Workspace* і носить ім'я имя, надане йому при створенні. Подвійний клік по файлу main.cpp призведе до того, що він відкриється в головному вікні.

Для закриття файла можна натиснути відповідний значок поруч з іменем файла, або використати клавіші **Ctrl+W**.

Для перемикання між відкритими файлами можна використати клавіші **Ctrl+Tab**, або натискати на їх заголовки мишкою.

Якщо файл був змінений, то на його вкладці зліва від імені файла з’являється зірочка (рис. 12). Для збереження файла потрібно натиснути **Ctrl+S** або в панелі інструментів вибрати кнопку **Save**.

Рис. 12. Панель вкладок з відкритим файлом

Через меню **View (**рис. 13) можно керувати зовнішнім виглядом ICP Code::Blocks. Для відображення або приховання панелей інструментів потрібно зайти в меню **View->Toolbars** і відмітити відповідні панелі.

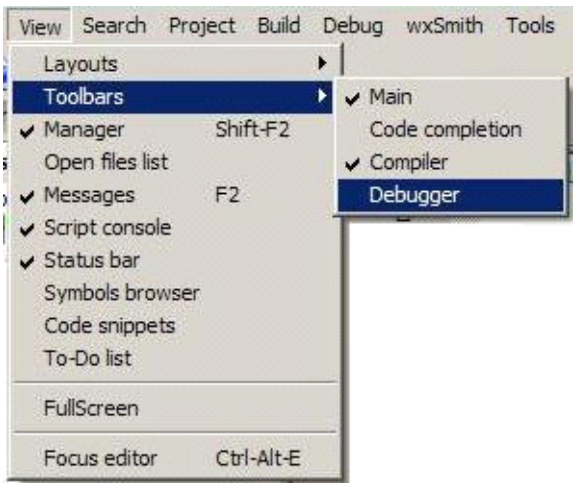


Рис. 13. Меню View

- **Main** – головна панель інструментів, на неї виводяться основні дії по роботі з проектами (рис. 14);

Рис. 14: Панель Main

- **Code completion** – панель для перегляду об’єктів кода (рис. 15);

Рис. 15: Панель Code completion

- **Compiler** – панель з кнопками керування компіляцією додатка (рис. 16);

Рис. 16: Панель Compiler

- **Debugger** – панель з кнопками керування відлагодженням додатку (рис. 17).

Рис.17: Панель Debugger

Пункт меню **View->Messages** або клавіша F2 відображають або приховають вікно повідомлень компілятора **Messages** внизу екрана.

Настройка кодування

Для нормальної роботи потрібно встановити кодування UTF-8 (див. рядок стану). Якщо встановлене інше кодування, то його потрібно змінити і після зміни перевідкрити всі відкриті файли.

Для настройки кодування Code::Blocks потрібно вибрати пункт меню **Settings->Editor...**, після чого відкриється вікно **Configure editor** налаштувань текстового редактора кода (рис. 18).

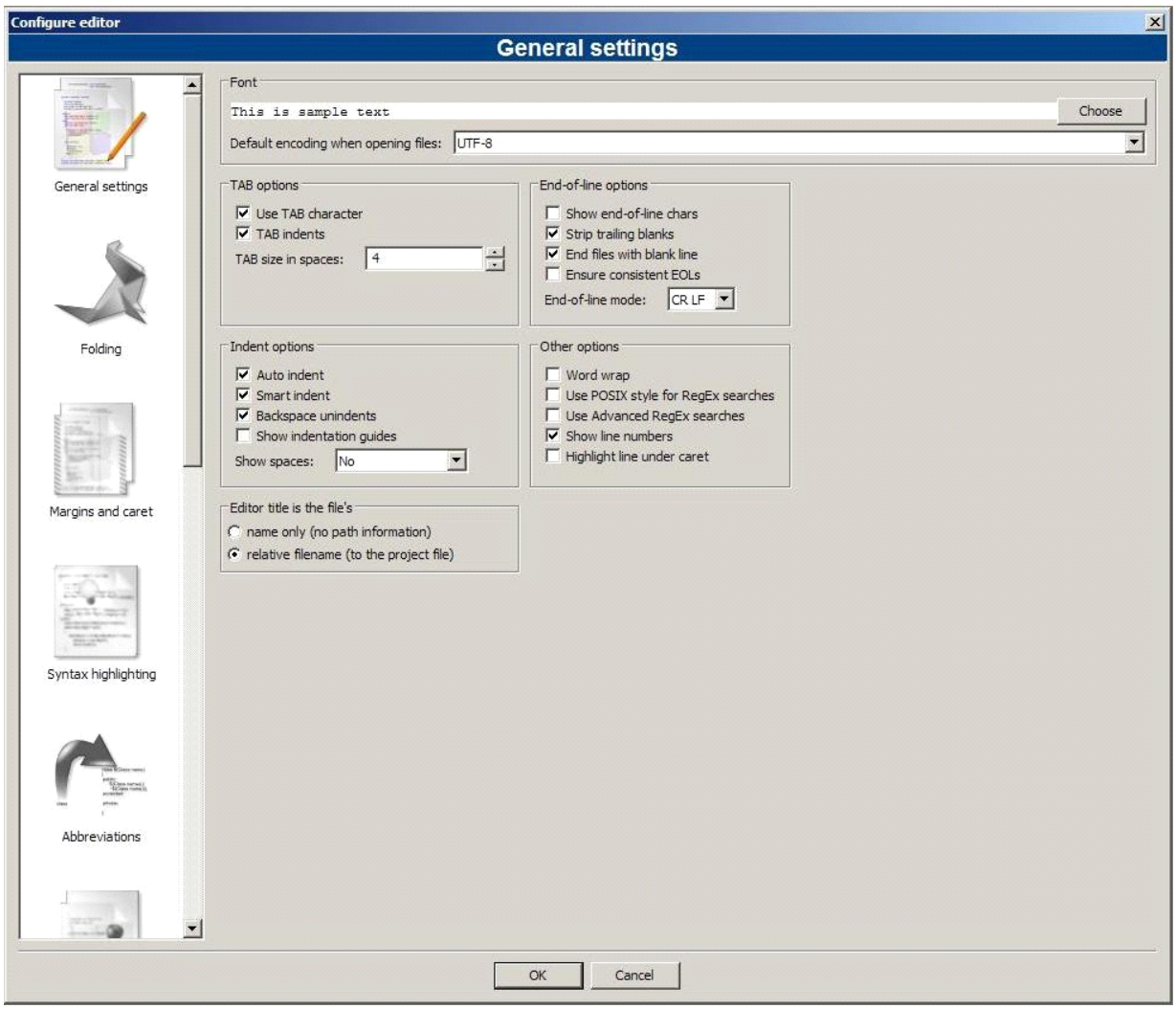


Рис. 18. Вікно налаштувань текстового редактора кода Configure editor

В випадаючому списку **Default encoding when opening files** потрібно вибрати кодування **UTF-8**, натиснути **OK** і перевідкрити всі відкриті файли в ICP.

Робота з декількома відкритими проектами

При одночасно відкритих декількох проектах, жирним шрифтом виділений активний проект, і незалежно від того, який файл в даний момент часу відкритий в основному вікні, компілюватися буде саме активний проект.

Для перемикання проектів потрібно навести мишку на той проект, який потрібно зробити активним, и натиснути праву кнопку мишки. В випадаючому меню (рис.19) обрати пункт **Activate project**. Для закриття цього проекту потрібно обрати пункт меню **Close project**.

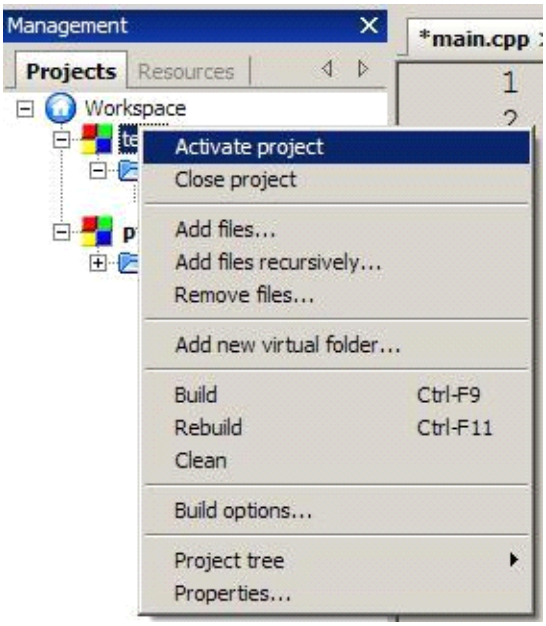


Рис. 19. Меню проекта

Компіляція и зборка проекта

Для зборки проекта, компіляції і запуска додатка потрібно натиснути **F9** або пункт в меню **Build->Build and run.**

- **Build** – зборка всього додатку
- **Complete current file** – компіляція файла, що відкритий в даний момент
- **Run** – запуск вже відкомпільованого додатку
- **Build and run** – зібрати і запустити
- **Rebuild** - перезібрати

- **Clean** – очистити проект від скомпільованих і тимчасових файлів.

Якщо потрібно тільки скомпілювати проект без запуску, то потрібно обрати пункт меню **Build** або натиснути **Ctrl+F9**.

При створенні проекта потрібно було визначити сценарії зборки додатку. Два сценарія зборки **Debug** або **Release** дозволяють отримати також два незалежних варіанти програми з різними опціями зборки, і відповідно, з різними опціями для оптимізації додатку. Для того, щоб використати їх, потрібно обрати відповідний сценарій зборки. Це можна зробити в панелі інструментів **Compiler** (рис.16), де можна задати один з двох сценарієв в випадаючому списку **Build target**. Змінити опції компіляції для кожного зі сценарієв можна у вікні **Project build options** (рис. 20) вибором відповідного сценарія **Debug** або **Release**.

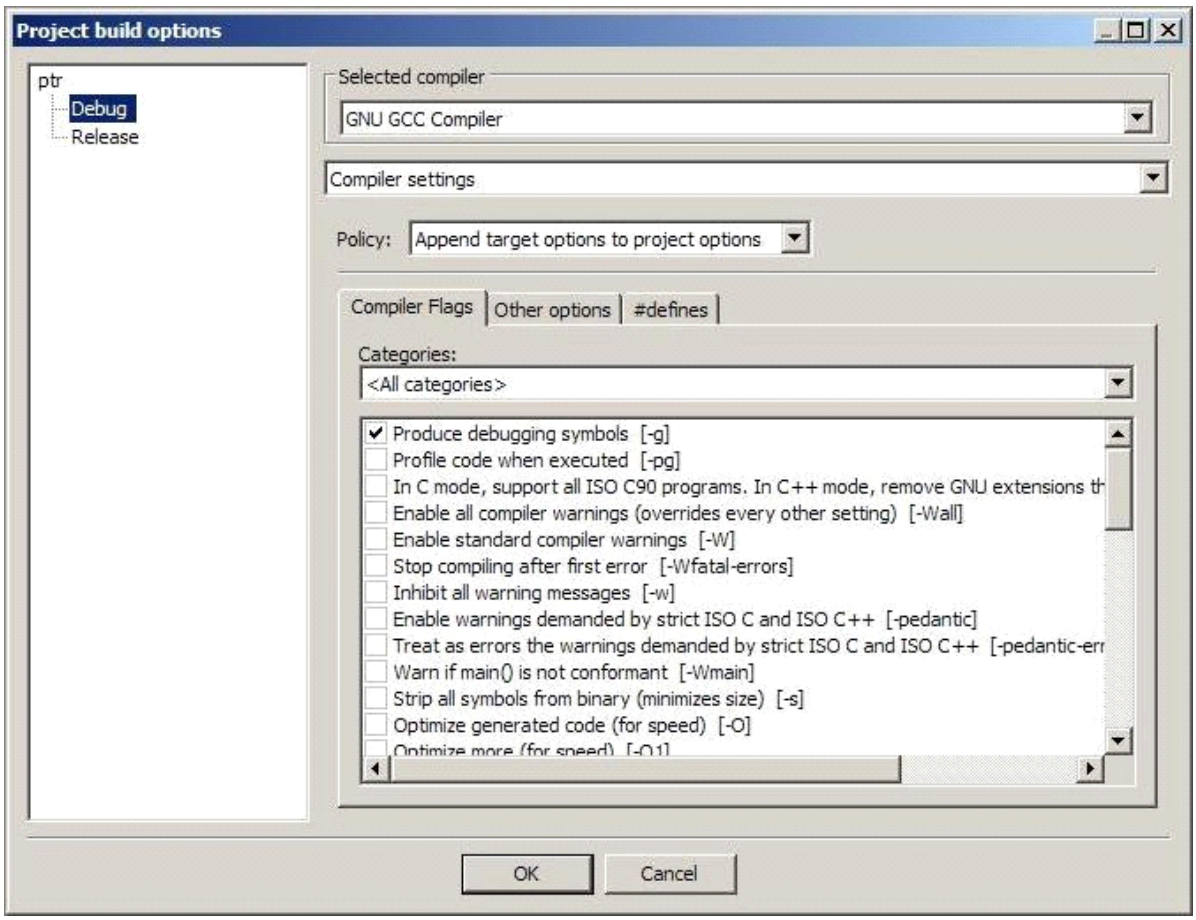


Рис. 20. Вікно налаштувань компіляції «Project build options»

Опції компілятора GCC

В певних випадках потрібне додаткове налаштування опцій компілятора. Ці налаштування проводяться для конкретного проекта у вікні **Project build options** (рис. 20), яке можна викликати з меню проекта вибором пункта **Build options...** (рис 19). Глобально ці налаштування задаються для всієї програми у вікні **Compiler and debugger settings** (рис.21).

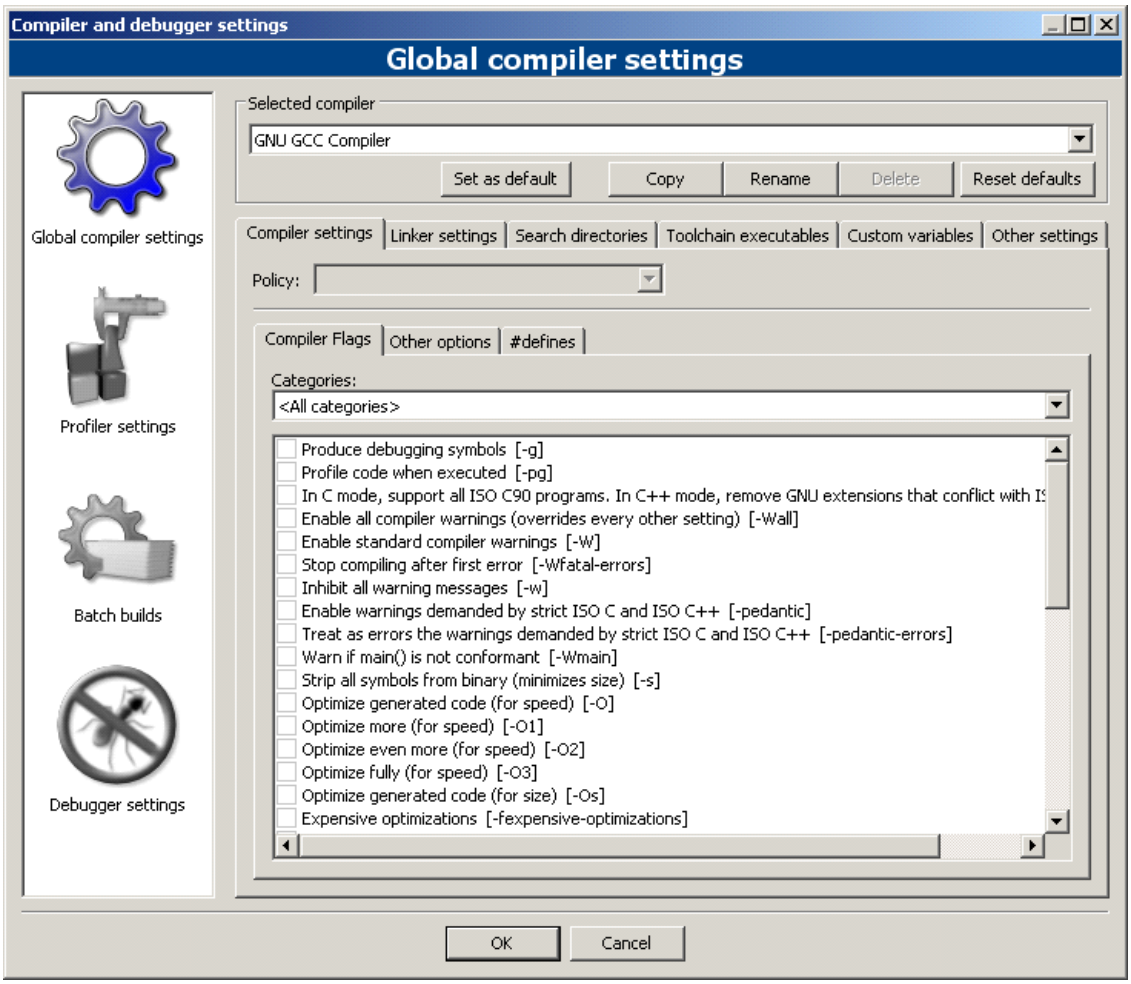


Рис. 21. Вікно налаштувань Compiler and debugger settings

Якщо програма не компілюється і в панелі **Messages** у вкладці **Build log** з'являється повідомлення «cc1plus.exe: error: unrecognized command line option „- Wfatal-errors‟», потрібно зайти в пункт меню **Settings -> Compiler and debugger...** і у вікні, що відкрилося, за умови, що обраний компілятор **GNU GCC Compiler**, у відповідному випадаючому списку **Selected compiler**, на

вкладці **Compiler Flags** потрібно зняти галочку з пункта «*Stop compiling after first error [-Wfatal- errors]*».

Якщо панелі **Messages** внизу екрана не видно, то для її відкриття потрібно натиснути **F2**.

Відлагодження проекта

Для пошуку помилок виконання в програмі використовується відладчик Gdb, який потрібно встановити. Для роботи з цим відладчиком використовується меню **Debug**. Для запуску відладчика потрібно обрати пункт меню **Debug->Start**, після чого запуститься відгадчик, і якщо не указати точки зупину, то він відпрацює по кроках весь додаток. Для перегляду вмісту змінних потрібно указати точки зупину. Точка зупину встановлюється або натисканням лівої кнопки миші на сірій розділювальній смузі поруч з номером рядка, або установкою курсора на рядку, в якому потрібно зупинитися.

Для перегляду вмісту змінних та масивів використовується панель (вікно) **Watches (**рис. 22).

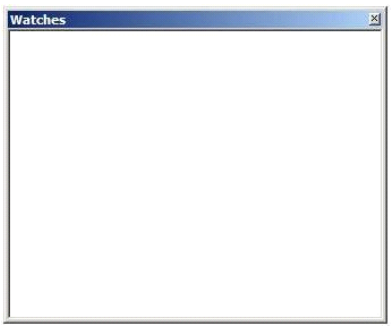


Рис. 22. Панель спостереження Watches

Для того, щоб додати в це вікно змінні, потрібно натиснути на вікні **Watches** правою кнопки мишки, після чого у випадаючому меню потрібно вибрати пункт **Add watch**.

Відкриється діалогове вікно **Edit watch**, де в полі **Keyword** потрібно ввести ім’я змінної, і натиснути кнопку **OK**. Для перегляду масива потрібно відмітити маркером **Watch as array**.

Додаток 3. ПЕРЕЛІК КЛЮЧОВИХ СЛІВ (стандарт С89)

auto	extern	signed
break	float	sizeof
case	for	static
char	goto	struct
const	if else	switch
continue	int	typedef
default	long	union
do	register	unsigned
double	return	void
enum	short	volatile
		while

Додаток 4. ОПЕРАЦІЇ В С

Унарні операції		
&	Взяття адреси	&x
*	Звернення за адресою (розіменування). Операндом мусить бути адреса	*&x
-	Унарний мінус	-1 -x
+	Унарний плюс	+a
~	Побітове заперечення (інверсія двійкового представлення)	int x=0; cout << ~x; // -1
!	Логічне заперечення (не), використ. для скалярних операндів.	!0=1 !1=0 !(7)=0 !!(0)=0

++	Інкремент.
	Не застосовується до констант або праводоп.виразів
	- префіксна форма: збільшення операнда до його використання
	- постфіксна форма: збільшення операнда після його використання
		++100 – невірно
		++(x*y*z)– невірно
		++i; // i=i+1
		j=++i;//i=i+1,j=i
		j=i++;// j=i,i=i+1
		i=++i + i++;
		// результат є системнозалежним, в мене при i=2 результат 7

--	Декремент. Див. ++	--j; // j=j-1
sizeof	Обчислення розміру операнда в байтах: Sizeof (вираз) Sizeof (тип)	 sizeof (1)=2 sizeof (char)=1
Бінарні операції		
Адитивні		
+	Бінарний плюс (до ариф. даних або вказівників)	x+y
-	Бінарний мінус (див.вище)	x-y
Мультиплікативні		
*	Множення (ариф.типи)	x*y
/	Ділення (ариф.типи). При цілочисельному діленні	 1/2=0

	результат округляється до найближчого цілого в сторону нуля	2/1=2 -4/3=-1
%	Залишок від цілочисельного ділення	1%2=1 3%2=1
Зсуву		
<<	Зсув вліво побітового представлення операнда зліва від << на кількість розрядів, що вказана справа від <<.	1<<2=4 0001 0100
>>	Зсув вправо побітового представлення операнда	4>>2=1 0100 0001
Порозрядні операції		
&	Порозрядна кон'юнкція (І) бітових представлень операндів	5&3=1 0101 0011 0001
|	Порозрядна диз'юнкція (АБО) бітових представлень операндів	5|3=7 0101 0011 0111
^	Порозрядне виключаюче АБО бітових представлень операндів	5^3=6 0101 0011 0110
Відношення		
<	Менше. Ариф.тип або вказівники. Результат - ціле число: 0 (хибне) і 1 (істинне)	1<2=1 2<1=0
>	Більше	1>2=0
<=	Менше-дорівнює	1<2=1
>=	Більше-дорівнює	1>2=0
==	Дорівнює	Мають нижчий приорітет, ніж інші операції відношення 1<2!=2>1, те саме, що і (1<2)!=(2>1) результат 0
!=	Не дорівнює	
Логічні операції Операції, що визначені для логічних даних. Логічні – false (0) і true (1, в С++ все, що не 0).		

&&	Бінарна кон’юнкція І (лог.множення)	1 приорітет
||	Бінарна диз’юнкція АБО (лог.дод.)	2 приорітет

Таблиця істинності:

A	B	Not A	A &&B	A ||B
False	False	True	False	False
False	True	True	False	True
True	False	False	False	True
True	True	False	True	True

Наприклад. Яким буде значення виразу
C=!(A&&B)||(!B&&B)&&B||A||!A&&(B&&B&&A)||B при A=0, B=1

(Відповідь C==1==true)

Операції присвоювання

Зліва може бути лише ліводопустимий вираз (наприклад, змінна, тобто ділянка пам’яті, в яку може буте вміщене певне значення)

=	Присвоїти значенню в лівій частині значення правої частини	x=2; y=2;
		y=x/2+x*2;
		x=y/2+y*2;
		//x=12, y=5

*=	S*=I+1 еквівал. S=S*(I+1)	
/=	S/=I еквівал. S=S/I	
%=	S%=S еквівал. S=S%S	
+=	S+=i еквівал. S=S+i	
-=	S-=i еквівал. S=S-i	
<<=	В ліву частину присв. значення, отримане зсувом на вказане число розрядів	X<<=2 екв. X=X <<2
>>=	Те саме	
&=	Z&=8 екв. Z=Z&8 (побітове І)	
|=	Z|=8 екв. Z=Z|8 (побітове АБО)	
^=	Z^=8 екв. Z=Z^8 (побітове виключаюче АБО)	

Операції вибору компонентів структурованого об’єкта

.	Прямий вибір компонента структурованого об’єкта	
->	Косвенний вибір структурованого об’єкта, що адресується вказівником.	

Операції з компонентами класів

.*	Пряме звернення до компонента	

	класа по імені об’єкта і вказівнику на об’єкт	
->*	Косвенне звернення до компонента класа через вказівник на об’ки і вказівник на компонент	
::	Операція указання області видимості	

Кома як операція

,	Декілька виразів, що розділені комами, обчислюються послідовно, зліва направо. Як результат приймається тип і значення най правішого виразу.	

( ) [ ] як операції

( ) [ ]	Виступають як операції при виклику функцій і індексації масивів	

Умовна операція (тернарна)

Вираз1 ? вираз 2 : вираз 3	Обчисл.вираз1. Якщо він істинний (не дорівнює 0), то обчисл. Вираз2, який стає результатом, якщо вираз1 хибний (==0), то результатом стає вираз3.	Визначення мінімального з двох: m=a<b?a:b

Операція явного приведення типів

(тип) операнд	Канонічна форма. Дозволяє привести операнд до потрібного типа. При приведенні більш широкого типу до вужчого можлива втрата точності.	(long)l (char) l (float) l
Тип (операнд)	Функціональна форма. Може використовуватися, коли тип має просте найменування.	char (l) – можна signed char (l) – не можна (В СВ можна) b=(char) a;
Операції new і delete		
new delete	Використовуються для динамічного розподілу пам’яті	new (p); delete (p)

Всі операції у виразі виконуються зліва направо згідно їх пріоритету. Порядок може бути змінений круглими дужками.

Пріорітет	**Операція**	**Асоціативність**
1.	() [] -> :: .	→
2.	! ~ + - ++ -- & * (тип) sizeof new delete тип( )	←
3.	.* ->*	→
4.	* / % (бінарні типу множення)	→
5.	+ - (бінарні типу додавання)	→
6.	<< >>	→
7.	< <= => >	→
8.	== !=	→
9.	&	→
10.	^	→
11.	|	→
12.	&&	→
13.	||	→
14.	?: (умовна операція)	←
15.	= *= /= %= += -= &= ^= |= <<= >>=	←
16.	, (кома)	→

Додаток 5. ЗАВДАННЯ НА ЛАБОРАТОРНІ РОБОТИ

ЛАБОРАТОРНА РОБОТА №1. ФАЙЛИ

Постановка задачі

В загальному випадку програма повинна складатися з трьох логічних блоків:

1) Послідовний доступ до елементів бінарного файла;
2) Прямий доступ до елементі бінарного файла;
3) Робота з текстовим файлом.

В межах кожного з логічних блоків реалізувати окрему задачу (табл.1)

Блок 1.

1. Сформувати бінарний файл f, компоненти якого обчислюються за формулою згідно варіанта.
2. Вказані по варіантах дії над компонентами файла (файлів) виконати в рамках функції, використовуючи послідовний підхід.
3. Вивести на екран значення елементів всіх сформованих в завданні файлів та результуючих файлів.

Зауваження.

1. Кількість компонентів файла f дорівнює n (n вибрати довільно в межах від 7 до 40).
2. Забороняється здійснювати прямий доступ до компонентів файлів.

Блок 2.

1. Сформувати файл (чи файли, якщо їх декілька за умовою), тип, кількість й взаємне розташування компонентів якого повинні відповідати умові завдання за варіантом.
2. Використовуючи прямий доступ до компонентів файлів, виконати завдання за варіантом.

Зауваження.

1. Обов'язковою є перевірка вхідного файла на порожність.
2. Використання додаткових файлів та масивів не допускається.

Блок 3.

1. За допомогою текстового редактора підготувати текстовий файл, який відповідає умові варіанта.
2. Виконати дії, зазначені в умові варіанта.

Зауваження.

1. Використовувати додаткові файли тільки в разі необхідності.
2. Обов'язковою є перевірка вхідного файла на порожність.
3. Словами називаються групи символів, відокремлені пробілами (одним чи кількома), і які не містять пробілів всередині.

Зміст звіту

1. Постановка задачі.
2. Текст програми.
3. Результати виконання програми на комп'ютері (вивести елементи вхідного та вихідного файлів).

Таблиця 1.

Варіант	***Завдання***
1	$\underbrace{f_i = \sqrt{2+\sqrt{2+\ldots+\sqrt{2}}}}_{i..коренів}$ Знайти найменше із значень компонентів файла f з парними номерами.
	Сформувати файл, компоненти якого є цілими ненульовими числами. Кількість від'ємних чисел дорівнює кількості додатних. Перекомпонувати файл так, щоб отримати послідовність чисел із чергуванням знаків.
	Дано текстовий файл. Дописати в кінець файла рядки з цього файла, що містять літери E, N, D, а також кількість рядків у вхідному файлі та номер останнього пустого рядку (якщо такого немає, то записати 0).
2	$f_i = \sqrt{3+\sqrt{6+\ldots+\sqrt{3\cdot i}}}$ Знайти найбільше із значень компонентів файла f по модулю з непарними номерами.
	Сформувати файл, компоненти якого є цілими ненульовими числами та розташовуються у наступному порядку: десять додатних, десять від'ємних і т. д. Кількість компонентів файла кратна 20. Перекомпонувати файл так, щоб розташування компонентів було наступним: 5 додатних, 5 від'ємних і т. д. (можна використати допоміжний масив).
	Підрахувати у текстовому файлі кількість літерних сполучень AB та ab. Дописати цю кількість в кінець файла словами.
3	$f_i = (ctg(i)) \cdot ln^i(i)$

	Знайти суму компонентів файла f, різницю першого і останнього компонентів.
	Сформувати файл дійсних чисел і виконати наступні дії: якщо компоненти файла впорядковані за неспаданням, то залишити його без змін. Інакше – перезаписати компоненти файла у зворотному порядку.
	Визначити у текстовому файлі співвідношення голосних, приголосних латинських літер та знаків пунктуації. Дописати в кінець файла рядки з отриманими значеннями.
4	$f_i = \cos(i) \cdot \log_2^i (i+1)$ Знайти найбільше із значень компонентів файла f.
	Сформувати файл дійсних чисел. Розширити файл, дописавши в його кінець свої ж компоненти, але в зворотному порядку.
	Підрахувати, скільки разів входить у текстовий файл кожна із наступних літер: A, B, C, D, E, F і вивести результат в текстовий файл у вигляді таблиці.
5	$f_i = i \cdot \cos^{i+1}(i+1)$ Знайти суму компонентів файла f, які розміщені на позиціях з парними номерами.
	Сформувати файл невпорядкованих від'ємних, нульових та додатних чисел. Переставити компоненти файла так, щоб спочатку розташовувались додатні, потім нульові, а за ними від'ємні компоненти, не змінюючи початкового взаємного розташування окремо додатних та окремо від'ємних компонентів. Наприклад, початковий файл: -1 0 5 -9 8 -3 5 0 -7 4 результат: 5 8 5 4 0 0 -1 -9 -3 -7

	Дано текстові файли f та g. Визначити, чи співпадають ці текстові файли. Якщо ні, то отримати координати (номер рядка і номер позиції у рядку) першого символу, починаючи з якого файли відрізняються. У випадку, коли один із файлів повторює початок іншого (довшого) файла - видати відповідне повідомлення.
6	$f_i = \prod_{k=1}^{i} \left(2^{k+1} - ln(k)\right)$ Записати в файл g із збереженням порядку слідування компоненти файла f, значення яких не перевищує 1000.
	Сформувати файл цілих чисел. Циклічно зсунути компоненти файла на n позицій вправо (n - натуральне і не перевищує кількості компонент файла).
	Дано текстові файли f та g. Записати до файла h таку початкову частину текстів у файлах f та g, що співпадає. У випадку відсутності такої частини видати повідомлення.
7	$f_i = \prod_{k=1}^{i} \sqrt{\sqrt{k} + \sqrt{k-1}}$ Сформувати файли f_1 і f_2. Формулу для формування f_2 взяти із наступного варіанта. Переписати із збереженням порядку слідування компоненти файла f_1 в файл f_2, а компоненти файла f_2 - в файл f_1, використовуючи проміжний файл h.
	Сформувати файл невпорядкованих чисел. Відсортувати компоненти файла по зростанню методом обміну.
	Вилучити з текстового файла f усі однолітерні слова і зайві пробіли. Результат записати до файла g.
8	$f_i = \sum_{k=1}^{i} \left(k^2 + 2^{k-1}\right)$

	Отримати в файлі g компоненти файла f, які є непарними числами.
	Сформувати файл, компонентами якого є латинські літери. Впорядкувати його компоненти у лексикографічному порядку.
	Знайти найдовше слово текстового файла f серед слів, друга літера яких N; якщо слів з найбільшою довжиною декілька, то знайти останнє. Якщо таких слів немає зовсім, то повідомити про це.
9	$f_i = log_3^{i+1}(i+2)$ Знайти добуток компонентів файла f, суму першого та передостаннього компонентів.
	Сформувати файл, компонентами якого є цілі ненульові числа. Кількість компонентів файла кратна 4. Кількість від'ємних чисел дорівнює кількості додатних. Перекомпонувати файл так, щоб числа розташовувались у наступному порядку: два додатні, два від'ємних і т.д..
	Визначити кількість слів у текстовому файлі та додати у кінець файла рядок, що містить перші літери кожного слова файла, записані у відповідній послідовності, причому кожна літера має бути унікальною.
10	$f_i = 2^{(i+1)} - (i-1)! + 1024$ Записати в файл g із збереженням порядку слідування компоненти файла f, перед якими розміщуються компоненти, кратні 3.
	Сформувати файл, компоненти якого є цілими ненульовими числами і розташовуються в наступному порядку: 5 від'ємних, 5 додатних і т.д. Кількість компонентів файла кратна 20.

	Перекомпонувати файл так, щоб розташування компонентів було наступним: 10 від'ємних, 10 додатних і т.д. (можна використати допоміжний масив).
	Текстовий файл f містить відомості про студентів курсу у наступному вигляді: прізвище ім'я по батькові, прізвище ім'я по батькові, …. Записати ці відомості до файла g за зразком: Ім'я По батькові Прізвище. Ім'я По батькові Прізвище. …
11	$f_i = i + \sum_{k=1}^{i} \left(2^k + 3^k\right)$ Записати в файл g всі парні числа файла f, а в файл h – всі непарні. Порядок слідування компонентів зберігається.
	Сформувати файл з n дійсних чисел. Переписати файл так, щоб його компоненти розташовувались в наступному порядку: $a_1, a_n, a_2, a_{n-1}, \ldots, a\left[\frac{n+1}{2}\right]$, де a_i – *i*-ий компонент файла.
	Вхідні дані такі ж, як у варіанті 10. Записати відомості до файла g за зразком: Прізвище І.П. Прізвище І.П. …
12	$f_i = \sum_{k=1}^{i} \frac{1}{k \cdot (k+1) \cdot (k+2)}$ Сформувати файли f і g. Формулу для формування файла g

	взяти із наступного варіанта. Записати в файл h спочатку компоненти файла f, а потім - компоненти файла g із збереженням порядку слідування.
	Сформувати файл невпорядкованих чисел. Відсортувати компоненти файла за неспаданням методом вибору.
	Дано текстовий файл f, що містить програму мовою С. Перевірити цю програму на невідповідність відкриваючих та закриваючих круглих дужок. Вважати, що кожний оператор програми займає не більше одного рядка файла f.
13	$f_i = \frac{i - 0.1}{i^3 + \lvert tg(2i) \rvert}$ Задане деяке число P. Переписати послідовно компоненти файла f в файл g, зупинившись після першого компонента файла f, який менший за P.
	Сформувати файл дійсних чисел. Циклічно зсунути компоненти файла на n позицій вліво (n- натуральне і не перевищує кількості компонент файла).
	Дано текстовий файл, що містить програму мовою С. Кожний рядок містить по одному оператору. Перевірити, чи стоїть у кінці кожного оператора крапка з комою, а наприкінці програми – закриваюча фігурна дужка. У відповідь видати процентне співвідношення кількості помилок до числа правильно записаних операторів.
14	$f_i = \sum_{k=1}^{i} \left(\frac{k-4}{2k!-1} \right)^{k+1}$ Знайти суму найбільшого і найменшого із значень компонентів файла f.

	Сформувати файл цілих чисел, кількість компонентів якого дорівнює 2n. Переписати файл так, щоб його компоненти розташовувалися в наступному порядку: a_1, a_{n+1}, a_2, a_{n+2},…, a_n,a_{2n}, де a_i – *i*-ий компонент файла.
	Дано довільне ключове слово і текстовий файл f. Сформувати файл g, що містить рядки файла f, символи у яких циклічно зміщені так, щоб ключове слово починалося з першої позиції рядка. Рядки, що не містять ключового слова, до файла g не записуються.
15	$f_i = \sum_{k=1}^{i} \frac{(-1)^k \cdot (2 + cos(2k))}{2^{k-1}}$ Знайти суму від'ємних компонентів файла f, суму модулів першого та передостаннього компонентів.
	Сформувати файл цілих чисел із чергуванням знаків, кількість компонентів якого кратна 10. Перекомпонувати файл, змінюючи порядок чисел всередині кожної десятки так, щоб спочатку йшли від'ємні числа десятки, а за ними - невід'ємні (зі збереженням вхідного порядку чисел).
	Дано текстовий файл f. Вилучити пробіли, що стоять на початку кожного рядка. Результат помістити до файла g. Послідовність рядків у файлі g повинна бути зворотною по відношенню до послідовності рядків файла f.
16	$f_i = i \cdot tg^{i-1}(i)$ Знайти найменше із значень компонентів файла f.
	Сформувати файл, компонентами якого є рядки довжиною не більше 10-ти символів, що складаються з латинських літер. Впорядкувати кожен компонент файла у лексикографічному порядку.

	У текстовому файлі подвоїти кожну цифру та вилучити усі латинські голосні літери.
17	$f_i = i \cdot sin^3(i)$ Знайти значення четвертого від кінця компоненту файла f.
	Сформувати файл дійсних чисел, кількість компонентів якого дорівнює 2n. Обчислити значення виразу: $(a_1 - a_{2n})(a_3 - a_{2n-2})\ldots(a_{2n-1} - a_2)$, де a_i – *i*-ий компонент файла.
	У текстовому файлі замінити кожну цифру на наступну за значенням цифру (9 замінити на 0), а перше слово кожного рядка переписати у зворотному порядку.
18	$f_i = \sum_{k=1}^{i} \frac{2^{k+1}}{\sqrt{k!}}$ Дано два файла f і g. Формулу для формування файла g взяти із наступного варіанта. Перевірити, чи еквівалентні файли f і g.
	Сформувати файл цілих чисел, кількість компонентів якого рівна 2n. Обчислити значення виразу: $a_1a_{2n} + a_2a_{2n-1} + \ldots + a_na_{n+1}$, де a_i – *i*-ий компонент файла.
	У текстовому файлі вилучити усі символи “+” і “-”, а також всі букви “б”, безпосередньо перед якими знаходиться буква “с”.
19	$f_i = \sum_{k=1}^{i} \left(\frac{k+1}{2k-1} \right)^k$ Знайти значення передостаннього компонента файла f та добуток першого та останнього компонентів.
	Сформувати файл з n цілих чисел. Обчислити значення виразу: $(a_1 + a_2 + 2a_n)(a_2 + a_3 + 2a_{n-1})\ldots(a_{n-1} + a_n + 2a_2)$, де a_i – *i*-ий компонент файла.
	Переписати вміст текстового файла f до файла g по рядках,

	додаючи в початок кожного рядка його порядковий номер (він повинен займати 4 позиції) і пробіл.
20	$f_i = \sum_{k=1}^{i} \frac{(-1)^k \cdot (k+1)}{k!}$ Знайти значення третього від кінця компонента файла f.
	Сформувати файл з n дійсних чисел. Обчислити значення виразу: $a_1 a_n + a_2 a_{n-1} + \ldots a_n a_1$, де a_i – *i*-ий компонент файла.
	Дано текстовий файл f і рядок S. Переписати до файла g усі рядки файла f, що містять у собі рядок S як фрагмент.
21	$f_i = \sqrt{i} + i!$ Сформувати файли f_1, f_2 , f_3, f_4. Формули для формування файлів f_2, f_3, f_4 взяти з інших варіантів. Організувати обмін компонентами між файлами у відповідності із схемою: $f_1 \rightarrow f_3$, $f_2 \rightarrow f_4$, $f_3 \rightarrow f_2$, $f_4 \rightarrow f_1$. Дозволяється використовувати лише один допоміжний файл.
	Сформувати символьний файл. Переставити в кінець файла його перший компонент, що є голосною літерою і стоїть між двома приголосними.
	Дано текстові файли f та g. Записати до файла h спочатку текст файла f, а потім текст файла g, пропускаючи усі пусті рядки.
22	$f_i = \sum_{k=1}^{i} \frac{1}{\sqrt{k!}}$ Перевірити, чи впорядковані компоненти файла f в порядку зростання.
	Сформувати файл цілих чисел. Переставити на початок файла 3 перші його компоненти, що є простими числами і не більші за 100. Порядок входження чисел не змінювати.

	Переписати із текстового файла f до файла g усі рядки, що містять більше ніж 60 символів.
23	$f_i = \sum_{k=1}^{i} (-1)^k \cdot 3^{k-1} - k!$ Знайти суму квадратів непарних компонентів файла f.
	Сформувати файл, що складається з рядків довжиною 10 символів. Поміняти місцями задані k-ий та p-ий компоненти файла.
	Переписати текст з файла f до файла g, замінюючи кожний символ “0” на символ “1” і навпаки. Вилучити рядки, у яких не зустрічається жодної цифри.
24	$f_i = \sum_{k=1}^{i} (-1)^{k+1} \cdot (k+1)! - 3^k$ Знайти кількість подвоєних непарних чисел серед компонентів файла f
	Сформувати символьний файл f. Переписати компоненти файла f у новий файл g у зворотному порядку, замінюючи всі пари підряд розміщених однакових літер на один символ ‘0’, якщо літери в парі були голосними, на ‘1’ – якщо приголосними і на ‘2’ – якщо символи не були літерами взагалі.
	Дано текстовий файл. Залишити у файлі лише перші входження кожного символу.
25	$f_i = \sum_{k=1}^{i} \frac{ln(k)}{k}$ Визначити модуль суми і квадрат добутку компонентів файла f.
	Сформувати файл, що складається з рядків довжиною не більше 20-ти символів. Замінити k-ий компонент файла на копію p-го компонента, записаного у зворотному порядку.

	Дано текстовий файл. Знайти найдовший рядок файла
26	$f_i = \sum_{k=1}^{i} \frac{k \cdot sh(k)}{2^{k-1}}$ Знайти добуток найбільшого і найменшого із значень компонентів файла f.
	Сформувати файл дійсних чисел. Поміняти місцями максимальний і мінімальний компоненти файла. Записати перед першим компонентом файла округлене значення середнього арифметичного всіх компонентів файла.
	Дано текстовий файл f. Переписати текст до файла g, вставляючи на початок кожного рядка стільки пробілів, скільки у ньому однолітерних слів.
27	$f_i = \sum_{k=1}^{i} \frac{k!}{\sum_{p=1}^{k} \left(\frac{1}{p+1} \right)}$ Отримати копію файла f в файлі g. Знайти добуток тих компонентів файла f, які за значенням більші свого номера.
	Сформувати файл, що складається з рядків, довжиною не більше 5 символів. Змінити порядок слідування компонентів файла на зворотній.
	Дано текстовий файл. Підрахувати кількість рядків, що починаються і закінчуються однаковою літерою.
28	$f_i = \prod_{k=1}^{i} \left(2 + \frac{1}{k!} \right)$ Знайти добуток компонентів файла f, які розміщенні на позиціях з непарними номерами.
	Сформувати файл, що складається з рядків довжиною не

	більше 15 символів. Вставити перед та після k-го компонента файла копію p-го компонента.
	Підрахувати у текстовому файлі кількість рядків, що складаються з однакових символів.
29	$f_i = \sum_{k=1}^{i} \frac{(-1)^k \cdot k \cdot (k+1)}{5}$ Число компонентів файла f повинно бути кратне 5. Записати в файл g спочатку найбільше значення перших п'яти компонентів файла f, потім – наступних п'яти і т.д.
	Сформувати файл цілих чисел. Визначити кількість компонентів файла, значення яких більше заданого числа M, та дописати знайдене значення на початок файла.
	У текстовому файлі замінити усі цифри символом "#" і вилучити усі пробіли.
30	$f_i = \sum_{k=1}^{i} \frac{(-1)^k \cdot ctg(k)}{(k-1)!}$ Знайти добуток додатних компонентів файла f та підрахувати кількість від'ємних компонентів.
	Сформувати символьний файл, що складається з літер та цифр. Перекомпонувати файл так, щоб всі цифри стояли на початку файла, зберігаючи вхідне взаємне розташування символів.
	Дано текстовий файл f. Переписати його до файла g, виключаючи пусті рядки та рядки, що містять одне слово. В інших рядках записати слова в алфавітному порядку.

Контрольні запитання

1. Поняття файла.
2. Структура текстового і бінарного файла. Відмінності між ними.
3. Відмінність і подібність масива і файла.
4. Класифікація файлів.
5. Операції над файлами.
6. Дії, які потрібно виконати для відкриття файла.
7. Визначення кількості компонентів у файлі.
8. Функції для запису та зчитування компонентів файла.
9. Розширення файла новими компонентами.
10. Закриття файла. Призначення і необхідність використання функції fclose( ).
11. Особливості текстових файлів. Доступ до компонентів текстового файла.
12. Особливості запису до текстового файла та читання з нього.
13. Функції, що використовуються для текстових файлів.
14. Обробка символів, що позначають кінець рядка в текстовому файлі.
15. Ознака кінця файла eof.
16. Переваги і недоліки файлів прямого доступу перед файлами послідовного доступу.
17. Оголошення файлів прямого доступу та файлів послідовного доступу.
18. Функції для обробки файлів прямого доступу.

ЛАБОРАТОРНА РОБОТА №2. НАБЛИЖЕНЕ ОБЧИСЛЕННЯ ІНТЕГРАЛА

Постановка задачі

Написати програму для обчислення з точністю $\varepsilon = 10^{-3}$ значення $I = \int_a^b f(x)dx$.

Вказівки до виконання завдання

Для обчислення значення визначеного інтегралу $I = \int_a^b f(x)dx$ з заданою точністю $\varepsilon > 0$ використати квадратурну формулу виду $I \approx \sum_{k=0}^{n} A_k f_k$, де $f_k = f(x_k)$, а x_k та A_k визначаються квадратурною формулою. Квадратурна формула, $f(x)$, a, b визначаються варіантом завдання (див. табл. 3).

Алгоритм квадратурної формули необхідно оформити у вигляді функції *з використанням функції як одного з параметрів.*

1. Для забезпечення необхідної точності при наближеному обчисленні інтеграла за квадратурною формулою необхідно обрати відповідне значення кроку h (чи, що те ж саме, число кроків N, на яке ділиться відрізок інтегрування). Для цього можна скористатися формулою залишкового члену квадратурної формули, однак ця оцінка значення відповідної похідної підінтегральної функції, яка входить до формули залишкового члену, нерідко викликає великі труднощі. Крім цього, цю оцінку доводиться робити для кожної нової підінтегральної функції.

На практиці для отримання бажаної точності обчислення за квадратурною формулою інтеграла часто використовується метод послідовного подвоєння кроків, який полягає у наступному.

Інтеграл I обчислюється за квадратурною формулою двічі: спочатку при кількості кроків, рівному N, а потім при кількості кроків, рівній $2N$. Похибка $\Delta\eta = |I - I_N|$ наближеного значення інтеграла I_n, обчисленого за

квадратурною формулою при кількості кроків, рівній n, обчислюється наближено за правилом Рунге: $\Delta_{2N} \approx \theta \left| I_N - I_{2N} \right|$. Для формул прямокутників і трапецій $\theta = \frac{1}{3}$, для формули Сімпсона $\theta = \frac{1}{16}$.

Таким чином, I_N обчислюється для послідовних значень $n = N_0$, $2N_0$, $4N_0$ і т.д., де N_0 - початкова кількість кроків. Процес обчислення закінчуються, коли для чергового значення n буде отримана похибка $\Delta_N < \varepsilon$. Початкову кількість кроків слід обирати від 10 до 50.

2. Для отримання достатньо ефективної програми необхідно врахувати, що у формулах трапецій і Сімпсона при подвоєнні кількості кроків немає необхідності обчислювати значення підінтегральної функції заново у всіх вузлах сітки, так як всі вузли сітки, отриманні при кількості кроків, рівній n, є вузлами сітки й при кількості кроків, рівній $2n$.

3. Для відлагодження функції інтегрування слід обирати такі підінтегральні функції та такі межі інтегрування, щоб відповідь була відома заздалегідь та обчислення інтеграла не потребувало великих затрат машинного часу. Необхідно потурбуватися про те, щоб для цієї функції насправді реалізовувався процес послідовного подвоєння числа кроків для отримання результату з заданою точністю.

4. Слід звернути увагу на те, що деякі помилки у програмі при реалізації метода подвоєння числа кроків (наприклад, врахування зайвого x_i) не впливають на точність результату, але ця точність досягається шляхом використання додаткового машинного часу.

Квадратурні формули.

1) формула прямокутників:

$$I = \int_a^b f(x)dx \approx h\left(f_0 + f_1 + \ldots + f_{N-1}\right),$$

де $f_i = f(x_i)$, $x_i = a + \left(i + \frac{1}{2}\right)h$, $h = \frac{b-a}{N}$;

залишковий член: $R = \frac{(b-a)^3}{24N^2} f''(\eta)$, $(a \le \eta \le b)$;

2) формула трапецій:

$$I = \int_a^b f(x)dx \approx h\left(\frac{1}{2}f_0 + f_1 + \ldots + f_{N-1} + \frac{1}{2}f_N\right),$$

де $f_i = f(x_i)$, $x_i = a + ih$, $h = \frac{b-a}{N}$;

залишковий член: $R = -\frac{(b-a)^3}{12N^2} f''(\eta)$, $(a \le \eta \le b)$;

3) формула Сімпсона:

$$I = \int_a^b f(x)dx \approx \frac{h}{3}(f_0 + 4f_1 + 2f_2 + 4f_3 + \ldots + 2f_{N-2} + 4f_{N-1} + f_N),$$

де $f_i = f(x_i)$, $x_i = a + ih$, $h = \frac{b-a}{N}$, N - парне;

залишковий член: $R = -\frac{(b-a)^5}{180N^4} f^{(IV)}(\eta)$, $(a \le \eta \le b)$.

Зміст звіту

1. Постановка задачі.
2. Текст програми.
3. Вхідні дані.
4. Результати тестів та розв’язку задачі.

Варіанти завдань

Функція *f(x)*, значення *a, b* та номер квадратурної формули

Таблиця 2.

№	f(x)	A	B	Формула
1	$\sqrt{e^x - 1}$	0,1	2	1
2	$e^x \sin x$	0	π	2
3	$(x^2 - 1)10^{-2x}$	0	1	3
4	$\ln\left(x\sqrt{1+x}\right)$	1	9	1
5	$\ln\left(\frac{1}{3+2\cos x}\right)$	0	π	2
6	$\sin\left(\frac{1}{x \ln^2 x}\right)$	2	3	3
7	$\frac{\arcsin\sqrt{x}}{\sqrt{x(1-x)}}$	0,2	0,3	1
8	$\cos\left(x^3 e^{2x} + \ln(x)\right)$	0	1	1
9	$\sin\left(\operatorname{tg}\left(\frac{x}{2} + \frac{\pi}{4}\right)\right)$	0	$\frac{\pi}{4}$	3
10	$e^{x \cdot \operatorname{arctg} x + \ln x}$	0	3	1
11	$\cos\left(\frac{1}{1+\sqrt{x}} + \sin x\right)$	0	4	2
12	$\ln\left(\frac{1}{5-3\cos x} + \frac{1}{x^2}\right)$	0	2π	3
13	$\operatorname{tg}\left(\frac{2}{1-4^x} + \ln\frac{1}{x}\right)$	1	2	1
14	$\frac{1}{(x+1)\sqrt{x^2+1}}$	0	$\frac{3}{4}$	2
15	$2{,}3^{\sin x + x + 0{,}2} \cdot \cos x$	0,1	1,8	3

16	$\sqrt{\ln x + tgx + 2{,}17} \cdot \sin x$	0,2	1,3	1
17	$e^{\cos x + \sqrt{x} + 0{,}318}$	0,2	1	2
18	$(1{,}32 \ln x + \arcsin x + 0{,}638) \cdot ctgx$	0,1	0,8	3
19	$arctgx \cdot \sqrt{\lg x + 0{,}532x + 1{,}175}$	0,1	1	1
20	$tgx \cdot \sqrt{0{,}378 \lg(4{,}56 + x) + x + 0{,}581}$	0,3	1	2
21	$ctgx \cdot \ln\left(\sqrt{\sin x + 1{,}23} + 0{,}83x^2\right)$	0,1	1,4	3
22	$\sin x \cdot \ln\left(\sqrt{x^2 + 0{,}162} + e^x + 0{,}187\right)$	0,2	1,1	1
23	$tgx \cdot \lg\left(\sqrt{\sin x} + 1{,}832x + 0{,}415^x\right)$	0,3	1,4	2
24	$arctgx \cdot \lg\left(0{,}618x + \sqrt{x + 0{,}21}\right)$	0,1	1	3
25	$\cos x \cdot \ln\left(x^2 + 0{,}872\sqrt{x} + 0{,}742\right)$	0,2	1,2	1
26	$arctgx \cdot 2{,}08^{\sqrt{x} + \sin x + 0{,}182}$	0,2	1	2
27	$tgx \cdot \sqrt{e^x \lg x + (x + 0{,}12)^2}$	0,3	1	3
28	$\ln\left(\sqrt{e^x + x} - 1\right)$	0,3	3	1
29	$\cos(x - \ln x)$	0	$\frac{\pi}{2}$	2
30	$x \cdot \sin x \cdot \ln x$	0,5	π	3

Контрольні запитання

1. Функція як параметр.
2. Вказівник на функцію.
3. Inline-функції.
4. Перевантажені функції.
5. Шаблони функцій.

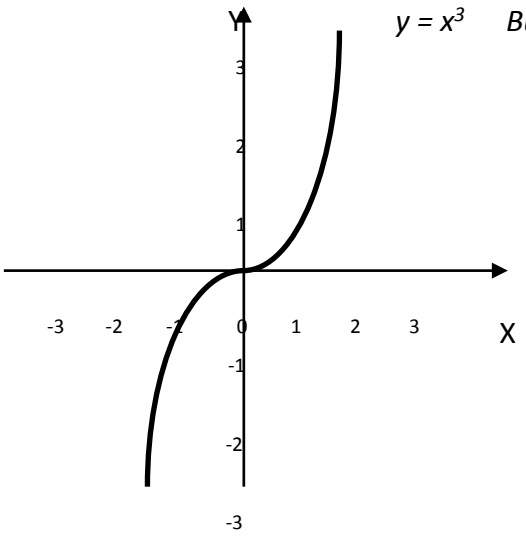
ЛАБОРАТОРНА РОБОТА № 3. ПОБУДОВА ГРАФІКА ФУНКЦІЇ

Постановка задачі

Використовуючи графічний режим, побудувати графік заданої функції, розв’язуючи наступні підзадачі:

1. Можливість зміни інтервалу обчислення значень функції (тобто кількість проміжків розбиття N задається константою).
2. Аналіз діапазону зміни значень функції на заданому проміжку зміни аргументу.
3. Зображення на екрані координатних осей X та Y з розміченою шкалою та проставленими числами, що відповідають діапазону зміни аргументу та функції.
4. Зображення на екрані графіка функції, формула якої визначається варіантом завдання.
5. Можливість масштабування виведеного зображення.
6. Зображення на вільній ділянці екрану математичної формули а також прізвища студента, що виконав завдання.

Можливий варіант виконання завдання зображено на мал.1.



Мал. 1

Зміст звіту

1. Постановка задачі.
2. Роздрукований текст програми.
3. Роздрукувати отриманий графік на різних інтервалах аргументу або для різних значень параметрів (2-4 різних випадки).

Вказівки до виконання завдання

Графік функції доцільно зображувати у вигляді ламаної лінії, що з'єднує точки, які належать графіку, тобто за допомогою частково-лінійної апроксимації кривої графіка.

Для побудови графіка по заданому рівнянню необхідно обчислити масив значень аргументу та масив відповідних значень функції. Крок зміни аргументу залежить від обраного діапазону зміни аргументу й кількості точок і обчислюється за формулою:

$dx = (x_{max} - x_{min})/(num - 1)$, де

dx - крок зміни аргументу;

x_{max}, x_{min} - відповідно максимальне та мінімальне значення аргументу;

num - обрана кількість точок графіку.

Паралельно з обчисленням масивів значень аргументу $(x[1...num])$ та функції $(y[1..num])$ необхідно визначити мінімальне й максимальне значення функції, які будуть потрібні для проведення масштабування.

Для зображення графіка на екрані обирається прямокутна ділянка, що визначається координатами лівого верхнього кута (x_{beg}, y_{beg}) і правого нижнього кута (x_{end}, y_{end}). При цьому потрібно мати на увазі, що лівіше й нижче поля виводу будуть знаходитися написи (відповідні значення аргументу і функції).

На наступному кроці необхідно визначити координати точок графіка, відображені в екранних точках (пікселях) з урахуванням розмірів обраного поля виводу графіка.

Масштабування по кожній з координатних осей відбувається згідно з формулами:

$mx = (x_{end} - x_{beg})/(x_{max} - x_{min})$ -масштаб по осі абсцис

$my = (y_{end} - y_{beg})/(y_{max} - y_{min})$ -масштаб по осі ординат.

Тоді координати *i*-тої точки графіку на екрані будуть визначатися згідно наступних формул:

$scrx[i] = x_0 + round(x[i] * mx)$

$scry[i] = y_0 + round(y[i] * my),$ де

$scrx[i], scry[i]$ - екранні координати *i*-ї точки;

$x[i], y[i]$ - значення аргументу і функції в *i*-й точці;

x_0, y_0 - екранні координати точки, що є початком координат для виводу функції;

mx, my - масштаби відповідно по осі абсцис і осі ординат.

На основі знайдених координат точок кривої графіка здійснюється зображення графіку на екрані шляхом з'єднання цих точок відрізками прямих. Потім виділяється поле виводу, здійснюється розмітка координат і виводяться відповідні написи.

Варіанти завдань

1. Парабола Нейля: $y = ax^{3/2}, x \in [0;5], a > 0$

2. Локон Аньєзі: $x^2 y = 4a^2(2a - y), x \in [-10;10], a > 0$

3. Конхоїда Нікомеда: $x = a + l\cos t \quad y = atgt + l\sin t$,

$t \in (-\frac{\pi}{2};\frac{\pi}{2}) \cup (\frac{\pi}{2};\frac{3\pi}{2}),\ l > 0,\ a > 0$

4. Цисоїда Діоклеса: $x = \frac{at^2}{1+t^2};\ y = \frac{at^3}{1+t^2};\ t \in (-\infty;\infty),\ a > 0$

5. Лемніската Бернуллі: $\rho^2 - a^2\cos 2\varphi = 0,\ a > 0,\ \varphi \in [0;2\pi)$

6. Строфоїда: $x = a(t^2 - 1)/(t^2 + 1),\ y = at(t^2 - 1)/(t^2 + 1),\ t \in (-\infty;\infty)$,

$a > 0$

7. "Хрест": $x^2 y^2 = a^2(x^2 + y^2)$ або $\rho = \frac{2a}{\sin 2\varphi},\ a > 0,\ \varphi \in [0;2\pi)$

8. Кардіоїда: $x = a\cos t(1 + \cos t),\ y = a\sin t(1 + \cos t),\ a > 0,\ t \in [0;2\pi)$

9. Астроїда: $x = a\cos^3 t,\ y = a\sin^3 t,\ t \in [0;2\pi),\ a > 0$

10. Декартів лист: $x = \frac{3at}{t^3 + 1},\ y = \frac{3at^2}{t^3 + 1},\ t \in [-10;-1) \cup (-1;10)$

11. Равлик Паскаля: $x = a\cos^2 t + b\cos t,\ y = a\cos t\sin t + b\sin t,\ a > 0,$

$t \in [0;2\pi)$

12. Ланцюгова лінія: $y = \frac{a}{2}(e^{\frac{x}{a}} + e^{-\frac{x}{a}}) = ach\frac{x}{a},\ a > 0,\ x \in [-2a;2a]$

13. Спіраль Архімеда: $x = at\cos t,\ a > 0,\quad y = at\sin t,\ t \in [\frac{\pi}{4};4\pi n + \frac{\pi}{4}]$

14. Параболічна спіраль: $\rho^2 = 2a\varphi,\ a > 0,\ \varphi \in [0;4\pi n],\ n = 1,2,3$

15. Логарифмічна спіраль: $b = ctg\alpha$; якщо $\alpha = \frac{\pi}{2},\ b = 0,\ \rho = ae^{b\varphi},\ a > 0,$

$\varphi \in [0;2\pi n],\ n = 1,2,3$

16. Циклоїда: $x = at - a_1\sin t, y = a - a_1\cos t$

розглянути випадки:

$a_1 < a$ - скорочена циклоїда

$a_1 = a$ - звичайна циклоїда

$a_1 > a$ - подовжена циклоїда

$a_1 > 0,\ a > 0,\ t \in [-2\pi;2\pi]$

17. Епіциклоїда: $x = (a+b)\cos t - a\cos((a+b)t/a)$

$$y = (a+b)\sin t - a\sin((a+b)t/a),\ a > 0,\ b > 0$$

Розглянути випадки:

1) якщо $\frac{b}{a}$ - ціле додатне число і $t \in [0;2\pi]$

2) якщо a та b - цілі додатні взаємо прості числа і $t \in [0;2\pi a)$

18. Гіпоциклоїда: $x = (b-a)\sin\frac{a}{b}t - a\sin\frac{b-a}{b}t$;

$$y = (b-a)\cos\frac{a}{b}t + a\cos\frac{b-a}{b}t,\ a > 0,\ b > 0$$

Розглянути ті ж самі випадки, що й у варіанті 17.

19. Трисектриса: $\rho = a(4\cos\varphi - \frac{1}{\cos\varphi}),\ a > 0,\ t \in [0;2\pi)$

20. Трактриса: $x = a(\cos t + \ln tg\frac{t}{2}),\ y = a\sin t,\ a > 0,\ t \in [-\frac{\pi}{2};0) \cup (0;\frac{\pi}{2}]$

21. Спірограф: Дано натуральні числа $A, B, D (D < B < A)$. Намалювати на екрані криву, параметрична форма якої має вигляд:

$x = (A-B)\cos t + D\cos\varphi$, $\quad y = (A-B)\sin t - D\sin\varphi$

де $\varphi = (\frac{A}{B})t$. Кут *t* змінюється від 0 до $2\pi n$, *n* дорівнює, *B* розділене на найбільший спільний дільник *B* та *A*.

22. Синус гіперболічний: $y = shx = \frac{e^x - e^{-x}}{2}$

23. Косинус гіперболічний: $y = chx = \frac{e^x + e^{-x}}{2}$

24. Арктангенс: $y = arctgx$

25. Еліпс: $x = r_1 \cos t$, $r_1 > 0, r_2 > 0$, $y = r_2 \sin t$, $t \in [0;2\pi)$

26. Синус: $y = \sin x;\ x \in [-\pi n; \pi n]$, $n = 1,2,3$

27. Тангенс: $y = tgx$, $x \in (-\frac{\pi}{2}; \frac{\pi}{2})$

28. Натуральний логарифм: $y = \ln x$

29. Експонента: $y = e^x$

30. Модуль косинуса: $y = |\cos x|$, $x \in [-\pi n; \pi n]$, $n = 1,2,3$

Контрольні запитання

1. Ініціалізація графічного режиму.
2. Графічні бібліотеки.
3. Система координат в графічному режимі.
4. Спосіб досягнення неперервності кривої графіка.
5. Функції для виведення тексту на екран в графічному режимі.
6. Шрифти для виводу тексту в графічному режимі.
7. Відмінність між штриховими і растровими шрифтами.
8. Опрацювання помилок графічного режиму.

ЛАБОРАТОРНА РОБОТА № 4. КЛАСИ І ОБ'ЄКТИ

Постановка задачі

Написати програму, в якій створюються і знищуються об'єкти, визначеніо користувачем; виконати виклики конструкторів і деструкторів.

Вказівки до виконання завдання

1. Визначити в програмі класс відповідно до варіанта.
2. Визначити в класі наступні конструктори: без параметрів, з параметрами, копіювання.
3. Визначити в класі деструктор.
4. Визначити в класі методи для перегляду і встановлення полів даних.
5. Визначити вказівник на метод.
6. Визначити вказівник на екземпляр класа.
7. Передбачити розміщення об'єктів як в статичній, так і в динамічній пам'яті, а також створення масивів об'єктів.
8. Написать програму, в якій створюються і знищуються об'єкти користувацього класа і кожний виклик констурктора і деструктора супроводжується видачею відповідного повідомлення (про те, який об'єкт який конструктор або деструктор викликав).
9. Показати в програмі використання вказівника на об'єкт і вказівника на метод.
10. Програма мусить використовувати три файла:
    - заголовочний h-файл з визначенням класа,
    - cpp-файл з реалізацією класа,
    - cpp-файл з демонстраційною програмою.

1. Приклад визначення класа:

```
const int LNAME=25;
```

```
class STUDENT{
char name[LNAME];          // ім’я
int age;                   // вік
float grade;               // рейтинг
public:
STUDENT();                 // конструктор без параметрів
STUDENT(char*,int,float);  // конструктор з параметрами
STUDENT(const STUDENT&); // конструктор копіювання
~STUDENT();
char * GetName() ;
int GetAge() const;
float GetGrade() const;
void SetName(char*);
void SetAge(int);
void SetGrade(float);
void Set(char*,int,float);
void Show(); };
```

2. Приклад реалізації конструктора з видачею повідомлення:

```
STUDENT::STUDENT(char*NAME,int AGE,float GRADE)
{
strcpy(name,NAME); age=AGE; grade=GRADE;
cout<< \nКонструктор з параметрами викликаний для об’єкта
<<this<<endl;
}
```

3. Конструктор копіювання викликається:

а) при використанні об’єкта для ініціалізації іншого об’єкта

```
STUDENT a(“Іванов”,19,50), b=a;
```

б) коли об’єкт передається функції по значенню

```
void View(STUDENT a){a.Show;}
```

в) при побудові тимчасового об’єкта як значення, що повертається функцією

```
STUDENT NoName(STUDENT & student)
{STUDENT temp(student);
temp.SetName(“NoName”);
return temp;}

STUDENT c=NoName(a);
```

4. Розміщення об’єктів в статичній і динамічній пам’яті:

а) масив студентів розміщується в статичній пам’яті

```
STUDENT gruppa[3];
gruppa[0].Set(“Іваненко”,19,50);
```

і т.д.

або

```
STUDENT gruppa[3]={STUDENT(“Іваненко ”,19,50),
        STUDENT(“Петренко”,18,25.5),
        STUDENT(“Сидоренко”,18,45.5)};
```

б) масив студентів розміщується в динамічній пам’яті

```
STUDENT *p;
p=new STUDENT [3];
p-> Set(“Іваненко ”,19,50);
```

і т.д.

Зміст звіту

1. Постановка задачі.
2. Текст програми та додаткових файлів з коментарями.
3. Вхідні дані.
4. Результати тестів та розв'язку задачі.

Варіанти завдань

(в варіантах задані описання лише методів класів користувача).

1. Студент: (ім'я – char*, курс – int, стать – int(bool)).
2. Службовець: (ім'я – char*, вік – int, робочий стаж – int).
3. Кадри: (ім'я – char*, номер цеха – int, розряд - int).
4. Виріб: (назва - char*, шифр – char*, кількість – int).
5. Бібліотека: (назва – char*, автор – char*, вартість – float).
6. Екзамен: (ім'я студента – char*, дата – int, оцінка – int).
7. Адреса: (назва міста - char*, вулиця - char*, номер будинка – int).
8. Товар: (назва – char*, кількість – int, вартість – float).
9. Квитанція: (номер – int, дата – int, сума – float).
10. Цех: (назва – char*, керівник - char*, кількість робітників – int).
11. Персонал: (ім'я - char*, вік – int, стать - int(bool)).
12. Автомобіль: (марка – char*, потужність – int, вартість – float).
13. Країна: (назва - char*, форма правління - char*, площа - float).
14. Тварина: (назва - char*, клас – char*, вид - char*,середня вага – int).
15. Корабель: (назва - char*, водотоннажність – int, тип – char*).
16. Речовина: (назва - char*, щільність – int, електропроводимість – char*).
17. Електродеталь: (назва - char*, потужність – float, вага - float).
18. Фігура: (назва - char*, кількість кутів – int, товщина ліній - float).
19. Студентська група: (назва - char*, курс – int, кількість студентів - int).
20. Паспорт: (ім'я- char*, вік – int, адреса – char*).

21. Оренда: (ім’я - char*, термін оренди – int, вартість оренди - float).

22. Записна книга: (ім’я власника - char*, поточний рік – int, кількість сторінок - int).

23. Час: (місяць – int, день – int, години – int, хвилини – int, секунди – int).

24. Розклад: (назва рейсу- char*, час відправлення - int, час в дорозі - int).

25. Будинок: (адреса - char*, кількість поверхів – int, рік забудови - int).

26. Інструмент: (назва - char*, матеріал - char*, призначення - char*, вага - float).

27. Архів: (місце знаходження - char*, кількість екземплярів – int, витрати на обслуговування - float).

28. Деталь: (назва - char*, призначення - char*, матеріал - char*, вартість - float).

29. Видання: (назва - char*, тематика - char*, номер – int, рік - int).

30. Столиця: (назва - char*, рік заснування - int, кількість жителів - int, бюджет - float).

Контрольні запитання

1. Основна різниця між процедурним та об’єктно-орієнтованим програмуванням.
2. Визначення об’єкта та класу.
3. Наслідування, інкапсуляція, поліморфізм.
4. Методи та особливості їх описання.
5. Доступ до властивостей і методів.
6. Конструктори, види конструкторів.
7. Віртуальні базові класи.
8. Бібліотеки та заголовочні файли.